Universidad Autónoma de Madrid
Escuela politécnica superior

Doble Grado en Ingeniería Informática y Matemáticas

Trabajo Fin de Grado

Diseño e implementación de un sistema para la recogida de logs en sistemas distribuidos

Autor: Mounaime Mellouk
Tutor: David Arroyo Guardeño
Ponente: Luis Lago Fernández

Junio 2016

Agradecimientos

Me gustaría agradecer a todas las personas que me han ayudado a hacer posible el desarrollo de este proyecto:

A mi tutor David por su gran ayuda y seguimiento durante todo el año.

A todos los profesores que me han ayudado y soportado con continuas tutorías a lo largo de mis cinco años de estudios.

A mi padre, por tu dedicación y trabajo para llevar la familia adelante, y por lo orgulloso que te sientes de mí y de mis hermanos. Gracias a ti estoy acabando mis estudios en España. Muchas gracias, papá.

A mi madre, por tu apoyo incondicional que siempre me has brindado en los momentos más difíciles, por darme la fortaleza de salir adelante, por tu preocupación constante de que no me falte de nada, y simplemente por darme ese amor que sólo las madres sois capaces de dar. No existen palabras en este mundo para agradecerte, mamá.

Resumen

Resumen

Un *log* es un fichero que contiene mensajes generados por el sistema o por un programa para registrar datos o información sobre quién, qué, cuándo, dónde y por qué un evento ocurre para un dispositivo o una aplicación en particular.

Los mensajes de los *logs* son muy importantes para los administradores de sistemas para entender el comportamiento previo del sistema. Pueden mostrar pistas de lo que está pasando en el sistema, en las redes, así como recolectar datos sobre el rendimiento y sobre todo la detección de intrusiones. Por lo tanto, son una gran fuente de información que permite determinar lo que ha pasado después de un incidente para poder tomar las medidas oportunas. Una vez una aplicación o sistema está configurado para la generación de *logs*, el siguiente paso es entender el significado de cada campo mediante el filtrado y normalización de los datos.

Sin embargo, uno de los problemas actuales es la heterogeneidad de los formatos de *logs*. Cada *log* tiene su propia estructura y sus propios campos, que no pueden ser extrapolados a otros ficheros *log*.

Es más, no toda la información contenida en el fichero es necesitada por el usuario, y a cada perfil de usuario de sistema le interesa una información distinta: es una tarea difícil identificar los campos útiles de entre todas las líneas, por lo que la conversión de un fichero *log* en un formato estándar se convierte en una tarea esencial para llevar a cabo.

El actual documento presenta una herramienta cuyo objetivo es ayudar a entender algunos ficheros *log*. Los ficheros tratados son aquellos de las distribuciones Linux, en particular SSH, Apache, Paquetes instalados, Fail2Ban y los cortafuegos. La herramienta usa una aplicación en Android para la visualización de resultados, y usa un servidor Flask en Python que es el encargado de *parsear* los *logs*, almacenarlos en un formato XML, y manejar los resultados para obtener estadísticas de los campos más importantes. Además, se le ha dado una especial importancia a los *logs* referentes a SSH y Apache, habiendo analizado los ataques por fuerza bruta y hecho *clustering* sobre las peticiones HTTP de Apache.

Palabras Clave

log, Linux, Formato, Heterogeneidad, *Parseo*, XML, SSH, Apache, Ataque por fuerza bruta.

Abstract

A log data, log message, or simply log, is what computer system device, software, etc. generates in response to some sort of stimuli. In other words, it is generated by some device to denote that something has happened.

Log messages are very important for system administrators to understand the previous system behaviour. Logs can show a lot of hints of what is happening on systems and network, from performance information to fault detection to intrusion detection, so, they are a good source of forensic information for determining what happened after an incident. Once a computer system is configured to generate log messages, the next step is to take in the meaning of each field, by filtering and normalizing the log data.

However, one of the current problems is the heterogeneity of format logs. Every log has his own structure and their own fields, and they can't be interpolated into other logs. Moreover, not all the lines of a log file are usually needed by an user: it is a hard task to identify information you need between all the data provided by a log file, so converting each log in a standard format become one of the essential task to carry out.

This paper presents a tool intended to be a help for understanding logs. The logs treated are those for Linux distributions, and specifically SSH, Apache, History packages, Fail2Ban and UFW. The app developed is on Android, and uses a Flask server on Python, which is the responsible for parsing logs, putting them in an XML format, and managing them to get some statistics of the more important fields. Besides, SSH and Apache have been given such a prominent place, analyzing brute force attacks and clustering Apache requests.

Keywords

Log, Linux, Format, Heterogeneity, Parse, XML, SSH, Apache, Brute Force Attack.

Índice general

Índice de Figuras

Índice de Tablas

1 Introducción

Los sistemas informáticos actuales están ejecutando una gran cantidad de programas continuamente. Además, con el auge de la red de Internet, la mayoría de los ordenadores alojan servidores para cualquier servicio. Todos estos programas están notificando las acciones que realizan los usuarios, en los ficheros de registro de actividad o ficheros *log*. No solo eso, sino que el propio sistema registra cada evento que sucede en el equipo, por lo que los *logs* se convierten en una fuente de información importante del estado del sistema, y el desarrollo de un sistema que recoja y analice de forma legible dichos *logs*, se convierte en un hecho de gran utilidad.

Los *logs* juegan un papel fundamental en la auditoría informática de sistemas, que es una pieza importante en el control del ciclo de vida de los sistemas [1]: la revisión y la evaluación de los controles, eficiencia y seguridad de los equipos se hace más eficaz con los archivos de registro de actividad. En particular, son de vital importancia para los sistemas de seguridad y seguimiento de eventos, esto es, permiten mantener un registro de las acciones dañinas de un atacante o intruso en el sistema.

Los administradores de sistemas están trabajando permanentemente con los ficheros *logs* para detectar las amenazas reales. Hacen inspecciones a nivel de tráfico de red, consolidan y analizan las alertas junto con los *logs* provenientes de sus redes, servidores, sistemas operativos y aplicaciones, dando como resultado cientos de miles de mensajes que posteriormente son reducidos a incidentes de seguridad [1]. Dichos incidentes son inmediatamente categorizados según su prioridad con base al impacto potencial de sus operaciones.

Es una tarea importante por lo tanto para el administrador del sistema elegir los ficheros *log* correctos para detectar anomalías, extraer la información adecuada, y sobre todo tener una visión global del sistema, haciendo uso de sistemas SIEM.

La tecnología SIEM (*Security Information and Event Management*)[2] proporciona un análisis de las alertas de seguridad generadas por el hardware y software de una red de computadores. Los sistemas SIEM desempeñan varias utilidades, desde la **agregación de datos**, es decir, la consolidación de datos desde varias fuentes, incluyendo redes, servidores y bases de datos, hasta la **correlación de los eventos de los *logs***, esto es, la búsqueda de los atributos comunes, y su relación con eventos de distintos *logs*.

El actual proyecto se centra en el sistema **Linux**, concretamente el sistema operativo **Ubuntu**. Hace una recogida de estadísticas de un conjunto de *logs*, junto con un análisis previo, y pretende ser un primer paso hacia una futura herramienta de procesamiento y detección de

anomalías.

1.1. Motivación del proyecto

La presente herramienta propone la recogida de la información más relevante de los *logs*, después de su previo análisis, mostrando los resultados en un dispositivo Móvil Android. De esta forma, la información condensada contenida en los *logs*, es tratada y presentada bajo una forma fácil y legible para el usuario final que, mediante una aplicación Móvil, puede ver a simple vista el comportamiento en el sistema.

El objetivo principal de la herramienta radica en proporcionar un entorno que haga factible la detección de comportamientos anómalos en un sistema. Esto es, se pretende construir una plataforma que permita integrar *logs* de fuentes diversas en una red, permitiendo así recopilar información sobre posibles problemas y casos de incidencias de seguridad en un sistema distribuido. Las estadísticas proporcionadas para cada *log* pueden llegar a ser claves para localizar aumentos de peticiones al sistema de algún tipo con fines dañinos.

Asimismo, el análisis de fuerza bruta realizado para SSH, clasificado por países y direcciones IP, es vital para detecciones de ataques por SSH. A lo anterior se le suma también la geolocalización, que persigue identificar vectores de ataque como los de norsemap (`http://map.norsecorp.com`), y que son de gran utilidad a la hora de detectar de modo visual ataques DDoS.

Por lo tanto, habiendo una gran diversidad de *logs*, todas las prestaciones anteriores motivan tener una herramienta capaz de sacar información importante de un conjunto grande y heterogéneo de ficheros *log*. Esto requiere el *parseo* de los *logs* y el almacenamiento en un formato legible de los datos extraídos. Finalmente, la correcta interpretación de esos datos puede beneficiarse de una herramienta de visualización de fácil uso.

En virtud de ese criterio de usabilidad, en este proyecto se ha puesto un especial énfasis en la modularidad de la aplicación diseñada y posteriormente implementada. En efecto, el software desarrollado corresponde a un primer enfoque y diseño de una herramienta para tratar el problema de recopilación, tratamiento y análisis de *logs* en sistemas distribuidos. De esta manera, dicha herramienta queda preparada para futuras mejoras y técnicas de análisis que se quieran añadir, y principalmente para tratar correlaciones entre distintos ficheros *log*.

1.2. Estructura del documento

El actual documento está dividido en **siete** capítulos.

El **primer** capítulo es introductorio, y pone el trabajo en contexto definiendo los objetivos que persigue así como la motivación para llevarlo a cabo.

En el **segundo** capítulo se discute el problema de recopilación y análisis de *logs*, mencionando las fases llevadas a cabo, las herramientas imprescindibles para ello, fundamentalmente las expresiones regulares, así como la posibilidad de seguridad y validación de los ficheros *log*. También se nombran algunas de las herramientas actuales con objetivo parecido al de la aplicación desarrollada.

El **tercer** capítulo consta del proceso de análisis previo realizado para el presente proyecto software. Se presentan y explican los *logs* que la aplicación analiza, junto a los fundamentos de

clustering y su validación Silhouette usados para las peticiones HTTP de Apache. Por otro lado, también se detalla el catálogo de requisitos de la aplicación.

Con el **cuarto** capítulo se pretende dar una visión del diseño de la aplicación, detallando las distintas interacciones entre los distintos módulos existentes con diagramas que muestran las distintas transiciones. Asimismo, se explican los paradigmas cliente servidor y modelo vista controlador que se han aplicado.

El **quinto** capítulo se centra en los detalles de la implementación de la herramienta, esto es, el entorno de desarrollo tecnológico empleado, los diferentes lenguajes de programación usados, así como las distintas clases Java y módulos Python creados para el correcto funcionamiento de la aplicación.

En el **sexto** capítulo se describe el plan de pruebas realizado detalladamente, tanto las pruebas unitarias como las de integración del sistema.

Por último, el **séptimo** capítulo, consta de las conclusiones del trabajo realizado, incluyendo algunas de las diferentes dificultades encontradas a la hora de llevarlo a cabo, así como las limitaciones del mismo y una serie de mejoras propuestas que quedan pendientes como trabajo futuro.

2

El problema de recopilación y análisis de *logs*

Los mensajes de *logs*, o simplemente *logs*, son un procedimiento de registro de actividad con objeto de poder trazar las actividades previas y el historial de acciones de los usuarios. Son habitualmente generados por un sistema de computación, un dispositivo o un software, como respuesta a algún tipo de estímulo.

Lo que es exactamente un estímulo depende fundamentalmente de la fuente que genera el *log* [3]. Por ejemplo, los sistemas Unix registran intentos de registro y de desconexión de usuarios en el sistema, los cortafuegos registran los mensajes *accept* y *deny* de las listas de control de usuarios ACL (*Access Control List*), los sistemas de almacenamiento de disco generan mensajes de *logs* cuando ocurre un fallo de escritura o lectura.

A pesar de su aparente diversidad, los *logs* se pueden clasificar en las siguientes categorías, según su propósito [1]:

- Informativos: Estos mensajes están diseñados para informar a los usuarios de que algo benigno ha ocurrido en el sistema, por ejemplo, Cisco IOS genera mensajes cuando el sistema se reinicia. Sin embargo son mensajes que pueden esconder información muy importante, ya que en este ejemplo se podrían detectar horas de inicio de sesión y de reinicio fuera del rango de actividad del usuario.
- Depuración: Los mensajes de depuración son generalmente generados por programas software para ayudar a los desarrolladores a la hora de identificar problemas en la ejecución de programas, por lo que todo programa software incluye un fichero *log*.
- Warning: Estos mensajes se centran en situaciones donde falta algo que tiene que estar en el sistema, sin embargo, su ausencia no tiene un impacto en las operaciones que el sistema lleva a cabo. Por ejemplo, si a un programa no se le incluyen los argumentos por línea de comandos, pero puede ejecutarse aún así, entonces el programa incluiría un mensaje *Warning* para el usuario.
- Error: Los mensajes *log* de error son usados para dejar constancia de los errores que ocurren a diferentes niveles del sistema. Por ejemplo, un sistema operativo debe generar un error en el *log* cuando no pueda sincronizar los buffers con el disco. Sin embargo, la mayoría de estos errores sólo dan información del punto de partida del error, por lo que hay que hacer un mayor esfuerzo de búsqueda para poder identificar la causa raíz del problema.

- Alerta: Las alertas detectan la ocurrencia de algo interesante en el sistema, generalmente en el ámbito de dispositivos de seguridad.

2.1. Fases de análisis de *logs*

En el mundo de la administración de sistemas, una gran parte del tiempo es empleada en revisar los *logs* para la detección de problemas potenciales. En el pasado, esta tarea era básicamente manual. El análisis de *logs* de sistemas actualmente suministra mecanismos automatizados para paliar con esta carga [1]:

- Recopilación de *logs*: Es el primer paso a llevar a cabo y consiste simplemente en la recogida de los ficheros *log* del sistema.

- Filtrado: En la etapa de filtrado se buscan los mensajes en los *logs* que se consideran importantes y que llevan información relevante. Aquella información considerada no interesante es ignorada para reducir la carga de procesado del sistema o herramienta. Una opción sería almacenar dicha información aparte para un posterior análisis de datos menos importantes.

- Normalización: En este paso se parte de un fichero de *logs*, se extraen de él varios campos (IPs de origen y destino, etc), y se ponen en un formato común. Este hecho es muy importante después para la etapa de correlación. Cuando un fichero *log* es normalizado, al resultado se le denomina conjunto de eventos. Otro paso importante en la normalización es la categorización. Esto significa transformar un mensaje contenido en un *log* en algo significativo, es decir, si por ejemplo hay un campo que sea "$ID = 6875$" y que representa el identificador de un *login* fallido, el campo en sí no es significativo, sin embargo hay que categorizarlo y es trabajo de la herramienta.

- Correlación: La correlación tiene como objetivo unificar eventos bajo etiquetas. Esto es, una situación en la que tenemos una conexión, intentos de registro de usuario fallidos, una aplicación desplegada en el sistema sin autorización, todo ello indica una posible infiltración en el sistema y abuso de los privilegios del sistema. Las dos herramientas básicas para la correlación son las reglas definidas y los tratamientos estadísticos.

- Acción: Es la acción que se lleva a cabo una vez se ha hecho correlación. La figura 2.1 muestra varias acciones posibles, desde el envío de correos y alertas, hasta el almacenamiento de los resultados del análisis para una mayor exploración posterior.

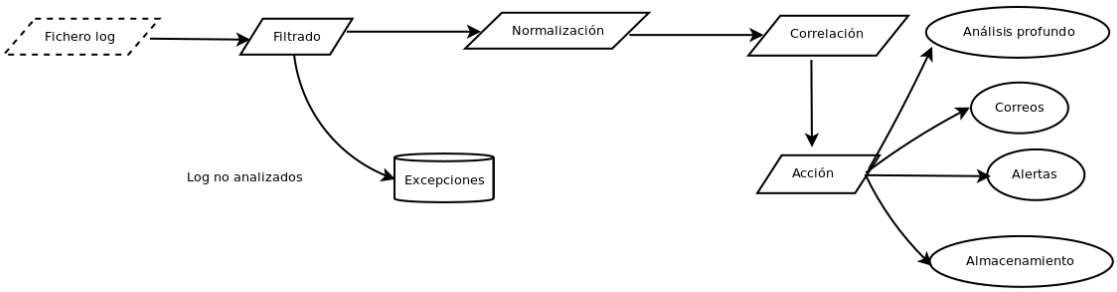


Figura 2.1: Proceso llevado a cabo para tratar un fichero *log*

2.2. Sistema de rotación de *logs*

Los *logs* o ficheros de registro están pensados para guardar la información de las actividades del sistema. Estos ficheros pueden crecer hasta llenar la partición donde se encuentran y bloquear el sistema, sin embargo, tienen muchísimos usos y son algo imprescindible, ya que aportan una inmensa cantidad de información de la actividad antigua del sistema.

En Unix, la herramienta *logrotate* es la encargada de rotar estos ficheros y ofrece más características interesantes para poderlos gestionar con facilidad y tener un control casi completo sobre ellos. Es importante conservar durante un tiempo bastante largo esta información, de ahí que se rotan los *logs*, ya que cada cierto tiempo, se creará un fichero sobre el que se escribirá y el que se tenía pasará a ser un histórico.

La ruta por defecto donde están los *logs* en las distribuciones Ubuntu de Linux es */var/log* .Hay distintos *logs* que están en la raíz de este directorio, sin embargo, también existen a su vez subdirectorios de otros servicios y aplicaciones, donde almacenan sus propios *logs* (Figura 2.2). Por ejemplo, en el subdirectorio *apache2*, se encuentran los *logs* de acceso y error del servidor Web Apache, en *apt* aquellos de los paquetes instalados del sistema, mientras que en *mysql* se hallan los del servidor de base de datos *MySql*.

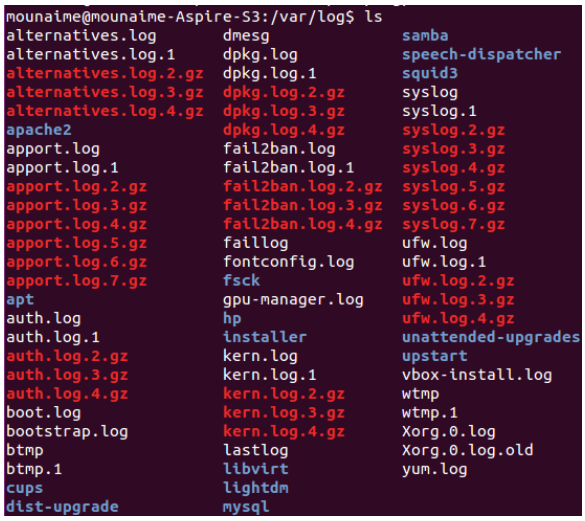


Figura 2.2: Directorio de *logs* en Ubuntu junto a un subdirectorio propio para algunas aplicaciones

En muchos sistemas se suele montar en una partición separada para que al llenarse un directorio, aunque no se puedan registrar más *logs* el sistema siga funcionando. En esa misma ruta los ficheros rotados tendrán una nomenclatura como:

$$log\ \ log.1\ \ log.2$$

en donde el primero que no tiene número es sobre el que estamos escribiendo actualmente, el siguiente *log*.1 es el anterior, y *log*.2 es el anterior a *log*.1, con lo que lo leemos del más reciente al más antiguo, tal y como muestra la figura 2.3.

El fichero de configuración principal en Linux es */etc/logrotate.conf*, en el cual se pueden configurar opciones de rotación de *logs* como las siguientes:

- Frecuencia de rotación: Por defecto, cada semana rotarán los *logs*, y la opción correspondiente es *weekly*. Se puede configurar para que sea diariamente (daily) o mensualmente (monthly).

- Tiempo de conservación de *logs* antiguos: Por defecto, los *logs* antiguos se guardan cuatro semanas, es decir, el campo *rotate* del fichero está a valor cuatro. Se puede configurar a petición del administrador.

- Compresión de ficheros: Con la opción *compress*, forzamos a que el sistema comprima los ficheros de registro para ahorrar espacio. Además, en la rotación se especifica cuándo se comienzan a comprimir los ficheros *log* resultando una extensión gz, que mediante el comando *zcat*, se pueden visualizar.

- Ejecutar comandos antes o después de rotaciones: Incluyendo las opciones *prerotate*/*postrotate*, se indica la ejecución de comandos o de *scripts* antes o después de las rotaciones de los *logs*.

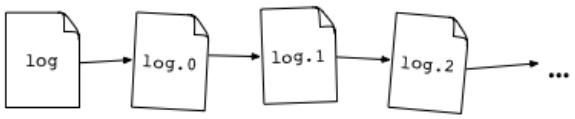


Figura 2.3: Encadenamiento de *logs* rotados

Fuente: http://www.nexolinux.com/logrotate-rotando-los-logs/

2.3. Expresiones regulares en el análisis de *logs*

Las expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto.

También denominados *regex*, constituyen un mecanismo bastante potente para realizar manipulaciones de cadenas de texto. El proceso para el que se usan estas expresiones, presente en el mundo UNIX y en lenguajes de programación como Python, es el de buscar y/o sustituir una subcadena de texto dentro de otra cadena. En principio esto puede hacerse usando métodos de *parseo* basados en comprobaciones, pero el problema surge cuando no tenemos una subcadena fija y concreta sino que queremos buscar un texto que responda a un cierto esquema. Por ejemplo: buscar aquellas palabras que comienzan con *http* : y finalizan con una }, o buscar palabras que contengan una serie de números consecutivos, etc. Es en estos casos cuando tenemos que utilizar las expresiones regulares.

Las expresiones regulares resultan de vital importancia en el problema de analizar y *parsear* los *logs*. En la mayoría de los casos, un fichero *log* contiene información relacionada con distintos eventos, y sin embargo, sólo estamos interesados en analizar y obtener información que tenga que ver con ciertos eventos en concreto, y que además sabemos los patrones que siguen, de ahí que busquemos dichas líneas usando las expresiones regulares.

Un ejemplo en Linux es el fichero *auth.log*, en el cual se registra información sobre autentificación, usuarios, y uso del comando *sudo*. Sin embargo, estamos interesados en este caso en obtener solamente la información relacionada con los intentos de acceso mediante SSH, por lo tanto no nos sirven muchas de las líneas del fichero. Además sabemos que dichas líneas son de la forma:

> Apr 24 19:17:01 Failed password for admin from 218.49.183.17 port 49468 ssh2
> Apr 24 19:17:02 Failed password for invalid admin from 218.49.183.17 port 49468 ssh2
> Apr 24 19:17:03 Accepted password for admin from 218.49.183.17 port 49468 ssh2

Por lo tanto, buscaremos el siguiente patrón:

DATE ACTION METHOD for (?: invalid user)? USER from IP port [0-9]+ ssh[0-9]

Donde:

```
ACTION = Failed|Accepted
IP =  ^(?:[0-9]{1,3}\.){3}[0-9]{1,3}
```

Mientras que METHOD y USER son simplemente una cadena de caracteres cualquiera.

Expresiones regulares en el contexto de seguridad

Las expresiones regulares tienen multitud de aplicaciones, y no sólo son importantes para el proceso de *parsear logs*, sino que también resultan de gran importancia en los temas relacionados con seguridad. Uno de los ataques contra el sistema basado en expresiones regulares es el llamado ataque ReDoS.

El ataque ReDoS es un ataque DoS (Denial of Service), es decir, que en caso de ser llevado a cabo, causa que un servicio o recurso sea inaccesible a los usuarios legítimos. El ataque explota el hecho de que la mayoría de implementaciones de expresiones regulares pueden alcanzar situaciones que causen su lento funcionamiento y rendimiento (exponencialmente proporcional al tamaño de la expresión regular).

El algoritmo que implementa las expresiones regulares está basado en un autómata finito no determinista (NFA), que es una máquina de estados donde para cada par de estado-entrada puede haber distintas posibilidades para elegir el siguiente estado. A la hora de evaluar una expresión regular, el motor de evaluación empieza a hacer transiciones como muestra la figura 2.4, y dado que puede haber muchas posibilidades, todas éstas son probadas para ver su concordancia o no con la expresión regular. El problema surge cuando las expresiones regulares son grandes, el autómata tiene varias posibilidades de exploración, y las opciones de evaluación se pueden hacer infinitas.

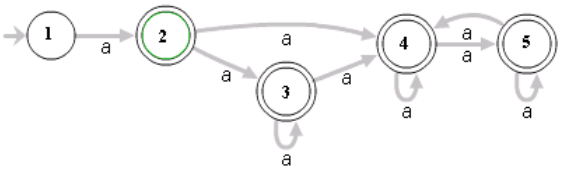


Figura 2.4: Autómata no determinista para una expresión regular

Fuente: https://www.owasp.org/index.php/Regular_expression_Denial_of_Service_-_ReDoS

Por lo tanto, el conocimiento de las expresiones regulares se convierte en una tarea vital tanto para el administrador como para el atacante. El atacante, con un buen conocimiento, puede ejecutar el ataque ReDos de forma eficaz, mientras que para el analista de seguridad, le ayuda a prevenir. Uno de las formas de prevención es el uso de un motor de evaluación de expresiones regulares eficiente y la limitación de la profundidad de la búsqueda a un cierto límite.

Además las expresiones regulares son habitualmente usadas también para crear patrones de detección y prevención de riegos en el sistema. Se pueden crear reglas basadas en expresiones regulares que permitan alertar y proteger contra ataques que intentan explotar sistemas sin parches o de difícil incorporación de parches. Por otra parte, también juegan un papel fundamental a la hora de validar campos en formularios para evitar la posibilidad de introducción de comandos o inyección de código.

2.4. Seguridad y protección de los *logs*

Los *logs* son una parte importante en la seguridad de cualquier sistema. Recopilan eventos ocurridos en el pasado tales como actividades de usuarios, estado de ejecución de programas, cambio en la información almacenada, etc.

Debido a su gran valor, los *logs* son un objetivo común para los atacantes de los sistemas. Un atacante que tiene acceso a un sistema, quiere naturalmente eliminar todas las pruebas y trazas de su presencia para esconder los detalles de su ataque. De hecho, el primer objetivo de un atacante experimentado es siempre el sistema de *logs*, por lo que la protección de dichos ficheros se convierte en una tarea fundamental.

Bruce Schneier, un criptógrafo experto en seguridad informática, expuso un método complejo basado en intercambio de claves para asegurar tanto la integridad como la validación y autentificación en los ficheros *log*.

El método parte de una máquina U, la cual no es físicamente segura para garantizar que no pueda ser invadida por atacantes. Sin embargo dicha máquina necesita ser capaz de mantener los ficheros *log* para un determinado programa o evento.

Con interacciones con una máquina segura, llamémosla τ, se quiere hacerlo posible, de tal forma que se garantice la integridad y autenticidad de los *logs* de U. En particular, no se quiere que un atacante que haya logrado tener acceso a U en tiempo t, sea capaz de de leer los *logs* generados antes del instante t, ni tampoco que sea capaz de alterar o borrarlos.

El sistema se basa en que la máquina sin seguridad U y que crea el fichero *log* inicialmente, comparte una clave secreta A_0 con una máquina de verificación segura. Cada vez que una entrada al *log* es añadida, la clave A_i se convierte en A_{i+1} mediante una función de sentido único. Todas las entradas del *log* se unen usando una cadena hash, por lo que cada entrada L_i contiene tres partes:

- El valor real de la entrada, M_i.
- Un elemento hash Y_i, calculado de la siguiente manera:

$$Y_i = H(M_i|Y_{i-1}), Y_0 = H(M_0)$$

- Un valor $Z_i = MAC_{A_i}(Y_i)$ calculado sobre el elemento Y_i, siendo $MAC_{A_i}(Y_i)$ el código simétrico de autentificación de Y_i bajo A_i.

Supongamos ahora que una máquina V quiere verificar un *log*. El esquema de validación sería el siguiente:

- V recibe una copia $[L_0, L_1, ..., L_f]$, donde f es el índice de la última entrada.
- Recorre todas las entradas calculando los valores hash Y_i y comprobando que coinciden.
- Posteriormente, V manda Y_f y Z_f a la máquina segura τ.
- τ conoce A_0, por lo que puede calcular A_f, lo que le puede comprobar el valor $Z_f = MAC_{A_f}(Y_f)$, e informar a V sobre el resultado de verificación.

Logs y privacidad de usuarios

La recolección de las actividades realizadas por el usuario con los *logs* es una actividad esencial para identificar asuntos relacionados con la seguridad de los sistemas, así como para mejorar sus servicios. Además, los métodos analíticos del Big data están siendo cada vez más usados en el mundo informático, y más concretamente en la detección de anomalías de seguridad. Fruto de esta captura y tratamiento continuo de información, aumenta la cantidad de información personal almacenada en los sistemas y se facilita el acceso a dicha información. Esta circunstancia puede, por consiguiente, vulnerar la privacidad de los usuarios.

En efecto, recolectar nombres de usuario y cualquier transacción asociada con estos usuarios puede representar un potencial ataque a su privacidad, y puede entrar en conflicto con legislaciones vigentes sobre la protección de datos personales [4].

No obstante, existen soluciones que se pueden llevar a cabo para conciliar la creación de registros de actividad (necesarios para un buen gobierno del ciclo de vida de los sistemas) y la protección de la privacidad de usuarios [4]. Por ejemplo, el **cifrado** de los ficheros *log* reduce el riesgo de accesos no autorizados. Asimismo, otro mecanismo de protección de la privacidad es la **anonimización** de los registros de actividad, la cual consiste en eliminar ciertas relaciones entre los datos de los *logs* con el fin de impedir la inferencia de posibles correspondencias entre acciones y usuarios.

2.5. Herramientas similares

La siguiente sección presenta un conjunto de herramientas similares a la desarrollada en el actual proyecto. Tienen básicamente como objetivo recopilar información relevante de los *logs*, presentando algunas estadísticas, usando algunos algoritmos y procesos de correlación entre la información obtenida, para poder notificar comportamientos extraños en el sistema a la vista de los resultados.

AI2

AI2 [5] es una plataforma desarrollada por investigadores del departamento de inteligencia artificial del MIT. Sus desarrolladores demostraron que predice ciberataques significativamente mejor que aquellos sistemas que dependen de humanos.

Se llegó a probar que AI2 puede detectar el 85 % de los ataques, un porcentaje que es tres veces mejor al que existía hasta el momento, mientras que también reduce el porcentaje de falsos positivos en un 5 %. El sistema se probó en cantidades de información que supera los 3.6 billones de GB, y que no era más que líneas de ficheros de *logs* de sistemas, generada por millones de usuarios durante un período de tres meses.

Para predecir ataques, AI2 analiza la información y detecta actividades anómalas mediante herramientas de clasificación de *clustering*, usando patrones de aprendizaje automático no supervisado.

GoAccess

GoAccess [6] (`https://goaccess.io/`) es una herramienta libre e interactiva para analizar *logs* de peticiones web en tiempo real, que se ejecuta en terminal para sistemas Unix. Suministra estadísticas HTTP rápidas para los administradores del sistema. La herramienta *parsea* los *logs* y muestra la salida en la terminal. Entre las estadísticas mostradas se encuentran las siguientes:

- Estadísticas generales: Número de peticiones válidas e inválidas, tipo de ficheros estáticos solicitados (css, gif, png, etc), ficheros no encontrados en el servidor (errores 404) y tamaño de los *logs* analizados, entre otras cosas.

- Geolocalización: Muestra la localización de las IPs que han hecho las peticiones, tanto el continente como el país.
- Códigos HTTP : Muestra los valores y el porcentaje de los códigos en las peticiones HTTP.

SEC

Simple Event Correlator [7] es una herramienta universal de proceso de eventos, que no sólo puede ser usada para ficheros *logs*, sino también para detección de fraudes. Está escrita en Perl y usa expresiones regulares para correlar los mensajes. Es capaz de leer de un fichero, asociar cada línea con un patrón usando la expresión regular correspondiente y después del proceso de correlación producir una salida, que generalmente es información mediante correo electrónico.

Graylog2

Graylog2 (https://www.graylog.org/) es un software libre de manipulación de *logs* muy completo. Almacena las estadísticas en *clusters* y grafica los resultados. La herramienta permite escribir expresiones regulares y en caso de haber entradas en los *logs* que las cumplan, mandar notificaciones en la aplicación.

Ofrece la posibilidad de eliminar información sensible como contraseñas, además de poder lanzar alarmas, mandar correos, o vincularse a *plugins* externos. De esta última forma los resultados de las estadísticas pueden ser reenviados hacia *plugins* externos. También permite hacer análisis sobre grupos de máquinas formando parte de una red.

Logstash

Logstash [8] es una herramienta completa para recogida de *logs*. Contiene desde opciones para recogida de *logs* locales, hasta almacenamiento de resultados en bases de datos e interacciones con *logs* desde bases de datos. A lo largo de sus versiones, ha ido añadiendo *plugins* para mejorar funcionalidad, entre algunos la posibilidad de conversión a formatos GELF, JSON, KV, XML, así como resolver direcciones IP en coordenadas geográficas o en nombres de hosts asociados. También Incluye búsquedas por *querys*.

Splunk

Splunk [9] suministra software para consolidar y analizar ficheros *log*, incluyendo aquellos no estructurados o de aplicaciones complejas. Permite recoger, guardar, indexar, buscar, correlar, visualizar y reportar todos los resultados para identificar y resolver los temas de seguridad lo más eficiente posible. Splunk también soporta ficheros a través de bases de datos, formato CSV, e incluso otros tipos de almacenamiento como pueden llagar a ser Hadoop o NoSQL.

Logtrust

Logtrust [10] es una herramienta que recolecta *logs* (infraestructuras, redes, aplicaciones y empresas) para crear un entorno de información seguro permitiendo a los clientes recibir alertas, hacer correlaciones, consultas y análisis de inteligencia en tiempo real. De esta forma, se puede abordar cualquier cuestión de seguridad, cumplimiento de normativas, monitorización y análisis de negocios en una única plataforma.

Provee una solución que permite manejar de manera segura los *logs* desde la nube, sin limitaciones en cuanto a origen de datos, espacio de almacenamiento ni ubicación geográfica.

PaperTrail

PaperTrail [11] es una plataforma de visualización instantánea de *logs*, así como una herramienta flexible para almacenar, representar estadísticas, y monitorizar los resultados de los análisis de *logs*. Por otra parte, es capaz de agrupar cada resultado dependiendo de su naturaleza, por ejemplo, errores en accesos a bases de datos, errores de peticiones HTTP o de aplicaciones,

asimismo como peticiones lentas ejecutadas en el motor de alguna base de datos, entre otras cosas.

3
Análisis

Esta sección trata del análisis realizado para llevar a cabo el proyecto.

Ante el actual problema de análisis y entendimiento de un conjunto de *logs*, se ha propuesto un proyecto cuyo principal enfoque es el diseño de una arquitectura base para posteriores mejoras e implementaciones de futuros *logs*, y así servir como base para futuros módulos de correlación.

Asimismo, se propone una interfaz cliente Móvil Android, con la principal misión de ofrecer al usuario usabilidad y facilidad a la hora de emplearla.

Por otra parte, después de un estudio previo, se ha hecho un análisis sobre un subconjunto de *logs* que se han considerado los más importantes en el sistema Linux, en cuanto a temas de seguridad se refiere: el *log* de SSH es de vital importancia para la detección de intentos de acceso no deseados al sistema, y junto con el de los cortafuegos, forman la base de la seguridad.

De vital importancia también es el *log* consistente de paquetes instalados: la instalación de paquetes sospechosos en horarios no habituales es síntoma de un atacante que tuvo la necesidad de instalar dichos paquetes para llevar a cabo su ataque. Asimismo, las peticiones HTTP son una parte fundamental en un sistema distribuido, y al ser un servicio que está en constante uso, se convierte en una fuente propensa de ejecución de ataques. Por lo tanto, es preciso analizar el contenido y las estadísticas del servidor web más utilizado en Linux: Apache.

3.1. *Logs* analizados

3.1.1. Apache

Apache es un servidor web HTTP de código abierto, para plataformas Unix, Microsoft Windows, Macintosh y otras, que implementa el protocolo HTTP/1.12.

Apache presenta entre otras características altamente configurables, bases de datos de autenticación y negociado de contenido. Además tiene amplia aceptación en la red, y desde 1996, Apache, es el servidor HTTP más usado.

En distribuciones Linux, el log de peticiones de Apache se encuentra en el directorio

$$/var/log/apache2/access.log$$

3.1.2. SSH

SSH (Secure SHell) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo el sistema mediante un intérprete de comandos. Además de la conexión a otros dispositivos, SSH permite copiar datos de forma segura, gestionar claves RSA y pasar los datos de cualquier otra aplicación por un canal seguro tunelizado mediante SSH.

En sistemas Ubuntu, la sintáxis básica del comando SSH es:

ssh usuario@direccionhost [comando]

Por ejemplo, se podría hacer un ls en la máquina remota y observar su salida mediante *ssh usuario1@direccionhost ls*

SCP

El comando scp es uno de los comandos más usados en SSH y permite copiar ficheros entre dos máquinas. Utiliza SSH para la transmisión de la información, por lo que ofrece la misma seguridad en las operaciones. Un ejemplo de uso sería el siguiente:

[usuario1@localhost]scp /tmp/file usuario1@servidor/tmp

Los mensajes relacionados con SSH se encuentran en el directorio */var/log/auth.log*, junto con mucha otra información de autentificación del sistema. Para cada intento de conexión SSH se registra la hora, el usuario, el host pedido, así como si la conexión tuvo o no éxito.

En la herramienta se recopilan las IPs con más conexiones, los usuarios más pedidos en las conexiones SSH, así como una geolocalización de IPs fallidas y una tabla de análisis con las IPs fallidas, número de intentos realizado, junto con la fecha y país de procedencia. La obtención de estos datos es de gran importancia, ya que permiten detectar anomalías y ataques, sobre todo aquellos de **Fuerza Bruta**, consistentes en mandar peticiones SSH con distintas contraseñas con la esperanza de acertar con alguna.

3.1.3. Paquetes instalados

El *log* history.log se encarga de almacenar los últimos paquetes instalados en el sistema, y se encuentra ubicado en el directorio */var/log/apt/history.log*.

Los paquetes instalados se dividen en bloques, teniendo cada paquete los siguientes campos:

- Fecha de instalación del paquete.
- Hora de instalación del paquete.
- Comando que se ha usado para la instalación del paquete.
- Nombre del paquete instalado.

Es importante verificar los paquetes instalados, así como las fechas y horas de instalación. Es una ayuda que puede facilitar información acerca de posibles intrusiones en el sistema. Muchos atacantes realizan instalaciones de paquetes necesarios para completar sus acciones, por lo que paquetes sospechosos u horas no habituales de instalaciones pueden hacer saltar la alarma.

En la herramienta se recopilan las estadísticas de las fechas, horas e instantes con más instalaciones, así como los últimos paquetes instalados.

3.1.4. Fail2Ban

Fail2ban es una aplicación escrita en Python para la prevención de intrusos en un sistema, que actúa penalizando o bloqueando las conexiones remotas que intentan accesos por fuerza bruta. Fail2ban busca en los registros de los programas que se especifiquen las reglas que el usuario decida para poder aplicar una penalización. La penalización puede ser bloquear la aplicación que ha fallado en un determinado puerto, bloquearla para todos los puertos, etc.

Actualmente Fail2Ban establece filtros para Apache, sshd, Qmail, vsftpd, Postfix entre otros. Los filtros son escritos con expresiones regulares de Python que establecen la regla que hará disparar una determinada acción sobre la IP que origina el hecho.

En sistemas Ubuntu, el fichero de configuración se encuentra en la ruta /etc/fail2ban/jail.conf. El archivo se organiza en secciones, donde se configura cómo se monitoriza un servicio en concreto. Los campos de cada sección contienen el nombre del servicio, el puerto, la localización del *log* del servicio, así como el número máximo de intentos antes de bloquear la IP. También se pueden añadir opciones de máximo tiempo de bloqueo y de envío de correos con el informe pertinente.

La siguiente sección es un ejemplo de configuración para el servicio SSH:

enabled = true

port = ssh

filter = sshd

logpath = /var/log/auth.log

maxretry = 6

bantime = 600

destemail = usuario@example.com

En cuanto al fichero *log*, se encuentra ubicado en Ubuntu en */var/log/fail2ban.log*. En la aplicación se han recogido las estadísticas de los campos más importantes: las IPs más bloqueadas, los servicios que más prohibición han obtenido, así como el número total de bloqueos de todos los servicios conjuntamente.

3.1.5. Cortafuegos

Un cortafuegos [12] es una combinación de hardware y software que aísla la red interna de la organización de Internet, permitiendo pasar a algunos paquetes y bloqueando a otros. Permite a un administrador de red controlar el acceso entre el mundo exterior y los recursos dentro de la red administrada. Entre los principios fundamentales de los cortafuegos se encuentran:

- Todo el tráfico que va del exterior hacia el interior de la red, y viceversa, pasa a través del cortafuegos, aunque hay muchas posibles configuraciones. La figura 3.1 muestra un cortafuegos que se encuentra en el límite de la red administrada y el resto de Internet. Las organizaciones de gran tamaño pueden utilizar varios niveles de cortafuegos.
- Sólo se permite el paso del tráfico autorizado de acuerdo con la política de seguridad local.
- El propio cortafuegos es inmune a la penetración. El mismo cortafuegos es un dispositivo conectado a la red, y si no está diseñado o instalado apropiadamente puede verse comprometido, en cuyo caso proporciona una falsa sensación de seguridad.

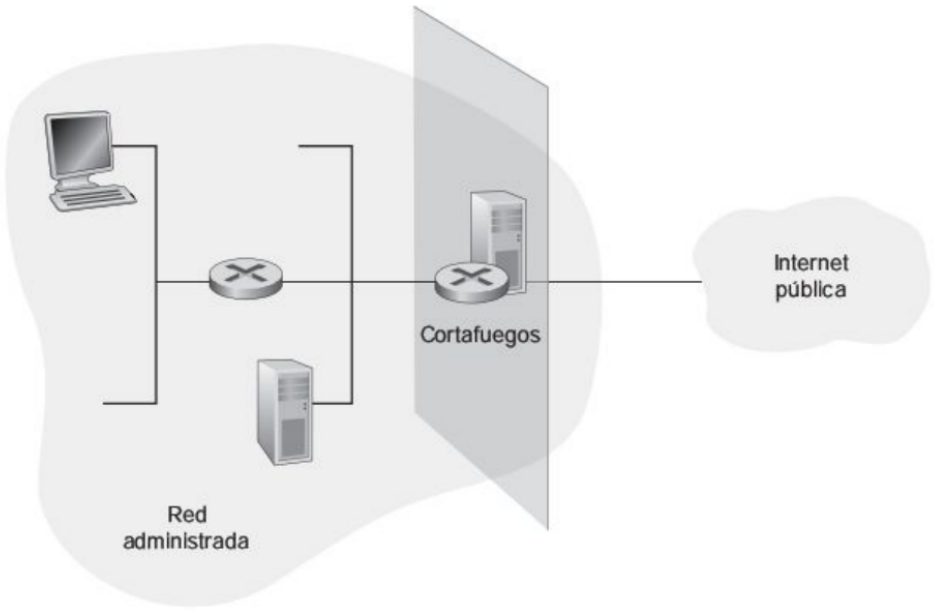


Figura 3.1: Colocación de un cortafuegos entre una red administrada y el mundo exterior. Imagen extraída de [12].

Los cortafuegos se pueden clasificar en distintas categorías [12]: pueden ser **filtros de paquetes tradicionales**, es decir el router de pasarela de una red actúa como filtro de paquetes para el tráfico que sale y entra de la red. También pueden ser **filtros con memoria de estado**, en cuyo caso las decisiones de filtrado no se toman para cada paquete de forma aislada, sino que se controlan las conexiones TCP y se utiliza dicha información para tomar las decisiones de filtrado. Por último, se encuentran aquellos que combinan los filtros de paquetes con pasarelas de aplicación.

Cortafuegos en sistemas Linux:

En sistemas Linux, los cortafuegos se instalan de la siguiente manera:

```
sudo apt-get install iptables
```

Este sistema de cortafuegos admite configuraciones, que se regulan mediante reglas, y que se dividen en tres tipos:

- *Entrada* (INPUT): Se usan para controlar las conexiones entrantes al sistema. Por ejemplo, intentos de acceso SSH. El sistema de cortafuegos mapea la dirección IP y el puerto con la regla asociada.
- *Enrutamiento* (FORWARD): Se usan para conexiones entrantes pero que en realidad no son entregadas localmente, como el caso de un router, al que le llegan paquetes constantemente para un posterior redireccionamiento.
- *Salida* (OUTPUT): Se usan para controlar las conexiones que salen del sistema. Por ejemplo, un intento de PING a una determinada dirección.

A su vez, una vez escogida una de las anteriores configuraciones, se puede determinar la accion a realizar, que es una de las siguientes:

- *Accept*: Acepta una determinada conexión. Por ejemplo la siguiente regla permite aceptar todas las conexiones procedentes de la IP 10.10.10.10:

  iptables -A INPUT -s 10.10.10.10 -j ACCEPT

- *Drop*: Elimina una conexión procedente de una determinada dirección. Actúa de tal manera que la parte que se conecta no se da cuenta de la existencia del cortafuegos y es como si no existiera. La siguiente regla permite bloquear todas las conexiones procedentes de la IP 10.10.10.10:

  iptables -A INPUT -s 10.10.10.10 -j DROP

- *Reject*: No permite la conexión procedente de una determinada dirección, pero manda un mensaje de error. Se usa en el caso de que se quiera rechazar una conexión pero se quiera informar de que el cortafuegos ha sido la causa de su rechazo. En este ejemplo se rechazan las conexiones SSH procedentes de la IP 10.10.10.10:

  iptables -A INPUT -p tcp –dport ssh -s 10.10.10.10 -j REJECT

Configuración alternativa de cortafuegos: UFW

En ocasiones, suele ser complicado entender todas las reglas de iptables que se suelen utilizar para configurar un firewall en Linux, sobre todo para usuarios con conocimiento básico del sistema operativo. Sin embargo, se puede usar toda la potencia de las iptables pero sin tener que conocer con exactitud todas sus reglas, usando los UFW.

UFW(Uncomplicated Firewall), es una interfaz de línea de comandos que consiste en un conjunto de órdenes simples que permiten configurar las iptables del firewall. A la hora de añadir una regla con UFW, existe la posibilidad de hacerlo de dos maneras distintas, o bien por el nombre del servicio (usando aquellos disponibles en /etc/services), o bien por identificación del puerto, IP, u otros parámetros.

Por ejemplo, para permitir conexiones ssh, se pueden usar cualquiera de los siguientes dos comandos:

sudo ufw allow ssh

sudo ufw allow 22

Asimismo, el borrado de reglas sigue el mismo patrón, cambiando el comando allow por delete. En el Anexo D se describen los comandos básicos de un *script* de configuración de un servidor Web usando UFW.

3.2. Análisis de peticiones HTTP

3.2.1. Necesidad y motivación

Para poder acceder a un sitio web, el servidor Web debe tener al menos el puerto 80 activo para procesar las peticiones HTTP. Sin embargo, los atacantes pueden modificar o manipular dichas peticiones y así causar problemas al servidor o tener acceso a información privilegiada del servidor. Usando peticiones HTTP, un atacante tiene una ruta legítima al servidor web, y por lo tanto puede fácilmente pasar cortafuegos y otras medidas de seguridad para iniciar el ataque.

En el ámbito de peticiones web HTTP, hay dos ataques que son los más comunes: **inyección SQL** [13] y **desbordamiento de buffer** [3].

- Inyección SQL: Es un método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar operaciones sobre una base de datos. Pongamos como ejemplo que se hace la siguiente petición HTTP:

  http://my.server.com/showResults.php?ID=5

  En este caso, suponiendo tener la tabla mytabledata a la que llama el método showResults, se ejecutaría la siguiente consulta SQL:

  SELECT * FROM mytabledata WHERE ID = '5'

  Sin embargo, un atacante puede modificar la petición HTTP, añadiendo simplemente una sentencia:

  http://my.server.com/showResults.php?ID=5 OR 1=1

  Ahora bien, la consulta que se ejecutaría en la base de datos sería la siguiente:

  SELECT * FROM mytabledata WHERE ID = '5' OR 1=1

  por lo que el atacante puede tener acceso a todas las entradas de la base de datos, incluso a información confidencial o sensible a la que no debería tener acceso.

- Desbordamiento de buffer: En términos generales, es un error de software que se produce cuando un programa no controla adecuadamente la cantidad de datos que se copian sobre un área de memoria reservada a tal efecto. Si dicha cantidad es superior a la capacidad preasignada, los bytes sobrantes se almacenan en zonas de memoria adyacentes, sobrescribiendo su contenido original, que probablemente pertenecían a datos o código almacenados en memoria, produciendo un fallo.

En HTTP, los atacantes mandan peticiones con longitudes largas, con el objetivo de provocar un desbordamiento de buffer cuando se procese la petición.

Por lo tanto, es necesaria una manera de detectar cuándo se están produciendo algunas de las situaciones anteriores, ya que pueden estar generando un peligro sustancial para el sistema.

3.2.2. Enfoque propuesto: Clustering

En este apartado se intenta dar una solución práctica para la **detección** de los dos problemas anteriores.

Para la inyección SQL, se propone la herramienta estadística de *clustering*, mientras que para el segundo problema, viendo la longitud de peticiones en las estadísticas de Apache de la aplicación, bastaría para detectar un intento de desbordamiento de buffer.

El *clustering* es uno de los métodos más usados y útiles para el procesamiento de información y división de datos en grupos, y así la identificación de distribuciones y patrones [14]. Su objetivo es particionar un conjunto de datos en grupos, llamados *clusters*, de tal forma que los puntos de un *cluster* sean más similares a los del propio *cluster* que a los del resto de *clusters*.

La similitud entre dos elementos de un *cluster* se define como una función distancia matemática, habiendo más similitud cuanto más pequeña es la distancia. Existe una gran variedad de distancias, siendo algunas de las más usadas la distancia Euclídea o la distancia de Manhattan [15], y dependiendo del problema, es conveniente usar una u otra. Sin embargo, en nuestro análisis, al ser los atributos (peticiones HTTP) cadenas, la distancia más adecuada es la distancia Levenshtein.

La distancia de Levenshtein [16] no es más que el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra, entendiéndose por operación, bien una inserción, eliminación o la sustitución de un carácter.

En el proceso de *clustering*, no hay clases predefinidas ni tampoco ejemplos que reflejen qué tipo de relaciones son las deseadas, y que serían las válidas para el tipo de datos que se tiene, por lo que se denomina un método de aprendizaje no supervisado [17]. Estos métodos son preferibles y tienen sentido cuando los datos no están etiquetados, siendo una solución óptima para analizar las peticiones Apache: a priori no se sabe qué tipo de peticiones se van a realizar, y los datos no están etiquetados de ninguna forma, lo que hace necesario un algoritmo de aprendizaje no supervisado para encontrar patrones.

En cuanto a los algoritmos de *clustering*, existen multitud de ellos en la literatura, sin embargo, se pueden clasificar principalmente en un conjunto de dos grupos [18]:

- ***Clustering* no jerárquico**: Son algoritmos de particionamiento que se basan en prefijar un factor k, y particionar los datos en k *clusters*. El algoritmo más usado en esta categoría es *k-means* [19], en el cual cada *cluster* tiene asociado un centroide (centro geométrico del *cluster*), y los puntos se asignan al *cluster* cuyo centroide esté más cerca, utilizando cualquier métrica de distancia. Iterativamente, se van actualizando los centroides en función de las asignaciones de puntos a *clusters*, hasta que los centroides dejen de cambiar.

  Otras variantes son GRASP [Greedy Randomized Adaptive Search Procedure] para evitar óptimos locales, *k-Modes* [20], que utiliza modas en vez de medias, y finalmente *k-Medoids* [20] que utiliza medianas para limitar la influencia de los *outliers*.

- ***Clustering* jerárquico**: Dependiendo del método que produce los *clusters*, se pueden dividir en dos:

  - **Algoritmos aglomerativos:** Producen una secuencia de esquemas descendientes de *clusters*. En cada paso, el esquema resultante es el resultado de fusionar en el paso anterior los dos *clusters* próximos en uno.
  - **Algoritmos divisivos:** Producen una secuencia de esquemas ascendientes de *clusters*. En cada paso, el esquema resultante es el resultado de dividir en el paso anterior un *cluster* en dos.

Para el análisis de las peticiones Apache, se ha usado *k-means*. Habitualmente se hace un gran número de peticiones a un servidor HTTP, eso supone elegir un algoritmo lo más eficiente posible. *k-means* es lineal en el número de datos, es decir, es de complejidad $O(n)$, mientras que los algoritmos de *clustering* jerárquico, al tener que tomar decisiones para fusionar y dividir *clusters*, son lentos y suelen ser de complejidad $O(n^2)$.

Se ha elegido para este proyecto un enfoque basado en *clustering* porque es con el que se tenía más familiaridad, sin embargo, existen multitud de técnicas que sirven para el análisis de aplicaciones Web, entre las cuales se encuentran las siguientes [21]:

- Cadenas de Markov: Las cadenas de Markov son procesos aleatorios consistentes en la transición de un estado a otro entre un número finito o contable de posibles estados. Han sido utilizadas para el estudio de detección de intrusos por autores como Zhang [22].

- Árboles de decisión: Son herramientas para análisis de múltiples variables y que pueden tomar decisiones de elección usando técnicas de grafos. Son comúnmente usados para la clasificación y pueden predecir la salida de nuevas observaciones. Una prueba del éxito de aplicar árboles de decisión en sistemas de detección de intrusos es el trabajo de Pfahringer [21].

- N-gramas: En el caso de detección de intrusiones, si un dato es considerado una cadena, entonces un n-grama es una subcadena de n caracteres. Han sido usadas ampliamente en el contexto de detección de intrusiones [23], desde 1-gramas hasta altos órdenes para n. Generalmente, el método se basa en contar el número de coincidencias de cada n-grama en una cadena. En el caso de tráfico Web, las cadenas son las peticiones HTTP.

3.2.3. Validación de *clusters*: Silhouette

Dado que el aprendizaje no supervisado no contiene clases predefinidas ni datos etiquetados, es necesario tener una forma de saber si la clasificación realizada es aceptable, y genera resultados útiles dado que la definición de los *clusters* no es conocida a priori [24], y se necesita algún tipo de evaluación.

Una buena clasificación consistiría en tener a todos los elementos de un mismo *cluster* cercanos, mientras que los *clusters* estén lejanos entre sí.

Rigurosamente, hay que minimizar la distancia intra-cluster (cohesión), maximizar la distancia inter-cluster (separación), y encontrar un método que tenga en cuenta las dos distancias para decidir la optimalidad de los resultados obtenidos.

El índice Silhouette es una medida de cohesión interna comparada con la separación entre los *clusters*. Dicho índice toma valores en el intervalo $[-1, 1]$, donde un mayor valor indica que el dato está bien clasificado en su propio *cluster* y por lo tanto si se cambia de *cluster*, estaría menos ajustado a dicho *cluster*.

Si la mayoría de los datos que se están clasificando tienen un mayor índice, eso indica que la clasificación general es la apropiada y la mejor dentro de las posibles. Asimismo, una gran proporción de datos con índice bajo cercano a -1 indica que la configuración del *clustering* no es la óptima.

El índice Silhouette puede calcularse con cualquier distancia métrica que use el *clustering*.

Formulación matemática:

Supongamos que un conjunto de datos ha sido agrupado usando *clustering* mediante cualquier técnica, pongamos $k-means$ por ejemplo, en k *clusters*. Para cada dato i, definimos $a(i)$ como el promedio de distinción de i de otros datos del mismo *cluster*. La interpretación de $a(i)$ es cómo de bien el dato i se ajusta a su *cluster*, de tal forma que valores pequeños indican mejor ajuste.

Definimos entonces el promedio de distinción de i a un *cluster* C como el promedio de distancias de i a todos los puntos del *cluster*:

$$a(i) = \frac{\sum_{j \in C} dist(i, j)}{|C|} \tag{3.1}$$

siendo C el *cluster* en el que se encuentra i.

Por otra parte, se define $b(i)$ como el promedio de distinción de i a cualquier otro *cluster* que no sea aquel al que pertenece i. El *cluster* con dicho promedio más bajo se denomina cluster

vecino, debido a que es el que mejor ajusta a i después del *cluster* al que pertenece actualmente el dato i. Por lo tanto se puede definir de la siguiente manera:

$$b(i) = \min_{C \in \mathscr{C}, i \notin C} a(i, C) \tag{3.2}$$

donde

$$a(i, C) = \frac{\sum_{j \in C} dist(i, j)}{|C|} \tag{3.3}$$

Teniendo los valores $a(i)$ y $b(i)$, es el momento de calcular $s(i)$, que es el índice Silhouette como sigue:

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \tag{3.4}$$

Que es equivalente a la siguiente fórmula:

$$s(i) = \begin{cases} 1 - \frac{a(i)}{b(i)}, & si\ a(i) < b(i) \\ 0, & si\ a(i) = b(i) \\ \frac{b(i)}{a(i)} - 1, & si\ a(i) > b(i) \end{cases} \tag{3.5}$$

Queda claro por lo tanto que $-1 \leq s(i) \leq 1$. Ahora veamos los posibles casos:

- $s(i) \approx 1$

  Para que $s(i)$ esté próximo a 1, se necesita que $a(i) \ll b(i)$. Dado que $a(i)$ representa el ajuste del dato a su propio *cluster*, y un valor muy grande de $b(i)$ indica el mal ajuste del dato a otros *clusters*, un valor próximo a 1 de $s(i)$ indica un buen ajuste del dato a su *cluster*, y por lo tanto interesan valores próximos a 1.

- $s(i) \approx -1$

  Para que $s(i)$ esté próximo a -1, se necesita que $b(i) \ll a(i)$ y por lo tanto se necesitan valores pequeños de $b(i)$, lo que indica que el dato se ajusta mejor a otros *clusters*. Por lo tanto un valor cercano a -1 indica un mal ajuste del dato a su *cluster*.

- $s(i) \approx 0$

  Un valor $s(i)$ cercano a 0 indica que el dato esta al borde de dos *clusters*.

  Por lo tanto, para un determinado *cluster* C, el promedio 3.6 de la medida $s(i)$ indica cuánto de bien se ajustan los datos al *cluster*:

$$\frac{\sum_{i \in C} s(i)}{|C|} \tag{3.6}$$

  Asimismo, si llamamos $\mathscr{C}$ al conjunto de *clusters*, tenemos que el promedio:

$$\frac{\sum_{C \in \mathscr{C}} \frac{\sum_{i \in C} s(i)}{|C|}}{|\mathscr{C}|} \tag{3.7}$$

representa el grado de ajuste total de los datos al *clustering*.

3.3. Catálogo de requisitos

A continuación se detallan el catálogo de requisitos de la aplicación, tanto los requisitos funcionales como los no funcionales.

3.3.1. Requisitos funcionales

RF 1 Para acceder a la aplicación el usuario deberá hacer *login* introduciendo su nombre de usuario y contraseña únicos.

RF 2 El usuario podrá salir de la aplicación en cualquier instante pulsando la opción salir del menú.

RF 3 Para cualquier *log* el usuario podrá cambiar en las preferencias el top de primeras estadísticas a mostrar.

RF 4 El usuario podrá ver las estadísticas relacionadas con el *log* Apache.

- **RF 4.1** Se podrá acceder al top de las IPs de peticiones HTTP.
- **RF 4.2** Se mostrarán los estados de HTTP más frecuentes de las peticiones a Apache.
- **RF 4.3** Se podrán consultar las longitudes que más se dan en las peticiones.

RF 5 El usuario podrá ver las estadísticas del *log* SSH.

- **RF 5.1** Se podrá acceder al top de las IPs conectadas por SSH.
- **RF 5.2** Se mostrarán los instantes más frecuentes de conexión.
- **RF 5.3** Se hará una recopilación de los usuarios más solicitados.

RF 6 El usuario podrá ver las estadísticas del *log* Fail2Ban.

RF 7 El usuario podrá ver las estadísticas del *log* history.

RF 8 El usuario podrá ver las estadísticas del *log* del cortafuegos.

RF 9 El administrador podrá ver el análisis relacionado con el *log* de SSH.

- **RF 9.1** Se desplegará en un mapa de Google Maps la geolocalización de las IPs de conexión más fallidas seleccionando la opción correspondiente.
- **RF 9.2** Se podrá visualizar una tabla resumen de las IPs fallidas, junto a la fecha, número de intentos de conexión y el país de procedencia.

RF 10 El administrador podrá ver los análisis del *clustering* de Apache.

- **RF 10.1** Se mostrará el número de *clusters* óptimo para los datos de peticiones.
- **RF 10.2** Se informará del número Silhouette asociado a la validación de *clusters* hallados.

3.3.2. Requisitos no funcionales

Seguridad

RN 1 Los datos entre la aplicación Móvil y el servidor Flask se envían empleando conexión segura SSL.

RN 2 La contraseña del administrador se almacena en la base de datos encriptada mediante la función criptográfica **MD5**.

RN 3 Es necesaria una identificación previa para poder acceder a los servicios de la aplicación.

Eficiencia y rendimiento

RN 4 Los *logs* menores o iguales a 1MB serán *parseados* en menos de 5 segundos.

RN 5 La aplicación mostrará la gráfica de resultados estadísticos en menos de 3 segundos.

RN 6 El número de *clusters* así como el índice Silhouette se calcularán en menos de 3 segundos.

RN 7 La geolocalización de peticiones SSH se mostrará en menos de 5 segundos una vez *parseados* los resultados.

Usabilidad e interfaz de usuario

RN 8 Las distintas pantallas de la aplicación Móvil siguen el mismo patrón gráfico.

Mantenimiento

RNF 9 La aplicación sigue una estructura modular por lo que el código tanto de la aplicación Móvil como el del servidor está dividido en distintos módulos. El caso de Java en paquetes siguiendo la directiva es.uam.eps.mellouk_tfg.

RNF 10 La aplicación está documentada con el objetivo de facilitar el mantenimiento posterior.

3.3.3. Casos de uso

Ver estadísticas de un campo de un *log*	
Actores	Usuario registrado
Descripción	El usuario navega en el menú de la aplicación eligiendo el *log* y el campo adecuados para así visualizar las estadísticas asociadas
Precondiciones	El usuario ha accedido correctamente a la aplicación. El usuario ha seleccionado el top de los primeros resultados en sus ajustes, N.
Postcondiciones	Se despliega una gráfica estadística con los primeros N valores del campo seleccionado

Cuadro 3.1: Caso uso correspondiente a ver las estadísticas de un campo.

Ver geolocalización IPs fallidas SSH	
Actores	Usuario registrado
Descripción	El usuario navega en el menú de la aplicación eligiendo la opción Análisis SSH y Geolocalización IPs
Precondiciones	El usuario ha accedido correctamente a la aplicación.
Postcondiciones	Se despliega un mapa de Google Maps con la localización señalada de los países con más intentos fallidos.

Cuadro 3.2: Caso uso correspondiente a ver geolocalización IPs fallidas.

Ver un *cluster* de peticiones a Apache	
Actores	Usuario registrado
Descripción	El usuario navega en el menú de la aplicación eligiendo la opción clustering Apache
Precondiciones	El usuario ha accedido correctamente a la aplicación. El usuario ha elegido el *cluster* correspondiente de entre la lista mostrada.
Postcondiciones	Se muestra una tabla con los contenidos del *cluster* elegido por el usuario

Cuadro 3.3: Caso uso correspondiente a visualizar *clusters* de peticiones Apache.

4
Diseño

En esta sección se detalla el diseño del sistema desarrollado, especificando los módulos y componentes que dan lugar al sistema y hacen posible su funcionamiento. Se ha realizado mediante esquemas y diagramas que facilitan y muestran las interacciones entre los distintos módulos.

4.1. Arquitectura del sistema

El sistema está desarrollado siguiendo el paradigma **cliente/servidor**. En este caso, el cliente Móvil **Android** realiza peticiones a la aplicación **Flask** corriendo en el servidor. De esta forma, al mostrar la aplicación Móvil cualquier estadística o análisis de cualquier *log*, ésta tiene que hacer peticiones HTTP al servidor para pedirle el recurso pertinente.

Por otro lado, el patrón de arquitectura software usado es el **Modelo-Vista-Controlador**. (MVC) que separa los datos y la lógica de negocio de la aplicación de la interfaz de usuario, y del módulo encargado de gestionar los eventos y las comunicaciones. De este modo, la separación de conceptos, facilita la tarea de desarrollo de la aplicación y su posterior mantenimiento.

En el caso de esta aplicación, los roles se dividirían de la siguiente manera:

- Modelo: Está integrado por los módulos del servidor, que proporcionan y representan la información de los *logs*, siendo así la lógica de la aplicación.
- Controlador: Lo forma la clase Java ServerInterface, que permite la conexión con el servidor, eligiendo la acción correcta y el recurso adecuado que se tiene que pedir al servidor. Así, el controlador invoca peticiones al modelo cuando se hace alguna solicitud sobre la información, y lo conecta con la vista.
- Vista: Está compuesta por la aplicación Móvil Android. Presenta la información en un formato adecuado para interactuar con el usuario.

Asimismo, el servidor está conectado a una base de datos, la cual le suministra los datos de autentificación del usuario, de tal forma que se ejecutan consultas a la base de datos a la hora de que el usuario se quiera identificar en la aplicación vía su dispositivo Móvil.

4.2. Entorno tecnológico

Los equipos en los cuales se ejecuta tanto la aplicación Móvil como el servidor tienen que verificar unos mínimos requisitos para el funcionamiento adecuado de la aplicación completa.

Para empezar, es necesaria una conexión a **Internet**, ya que la aplicación Móvil tiene que comunicarse con el servidor, que a priori no tiene que estar alojado en el mismo host que la aplicación. Sin embargo, en caso de que tanto la aplicación como el servidor estén en el mismo equipo o en una misma red local, no haría falta una conexión a Internet.

Por un lado, al ser el servidor implementado en Python, éste no requiere grandes requerimientos hardware, bastaría con un equipo de **32Mb** de RAM, aunque es aconsejable una mayor capacidad de memoria. Tampoco hay una limitación de procesador, aunque lo adecuado es un mínimo de **500MHz** para garantizar un tiempo de respuesta satisfactorio.

Por otro lado, para la ejecución de la aplicación Móvil, es necesario[1] tener un dispositivo con un mínimo de **32Mb** de RAM, **32Mb** de Memoria Flash, y **200MHz** de procesador.

4.3. Flujo de interacción entre módulos

La aplicación se compone fundamentalmente de cuatro bloques principales a distinguir. La aplicación Móvil Android, el servidor Flask, los módulos de *parseo*, y la base de datos de usuarios. A continuación se muestra la interacción entre ellos:

- **Aplicación-Servidor**

  Al pedir estadísticas de algún *log*, así como algún tipo de análisis SSH o Apache, la aplicación solicita los datos al servidor. Éste las devuelve en formato JSON, para que sean visualizados por la aplicación Android.

- **Servidor-Parsers** El servidor, mediante los módulos correspondientes, interactúa (ver figura 4.1) con los módulos de *parseo* de *logs*, a la hora de pedir información acerca de estadísticas o análisis de ciertos *logs*.

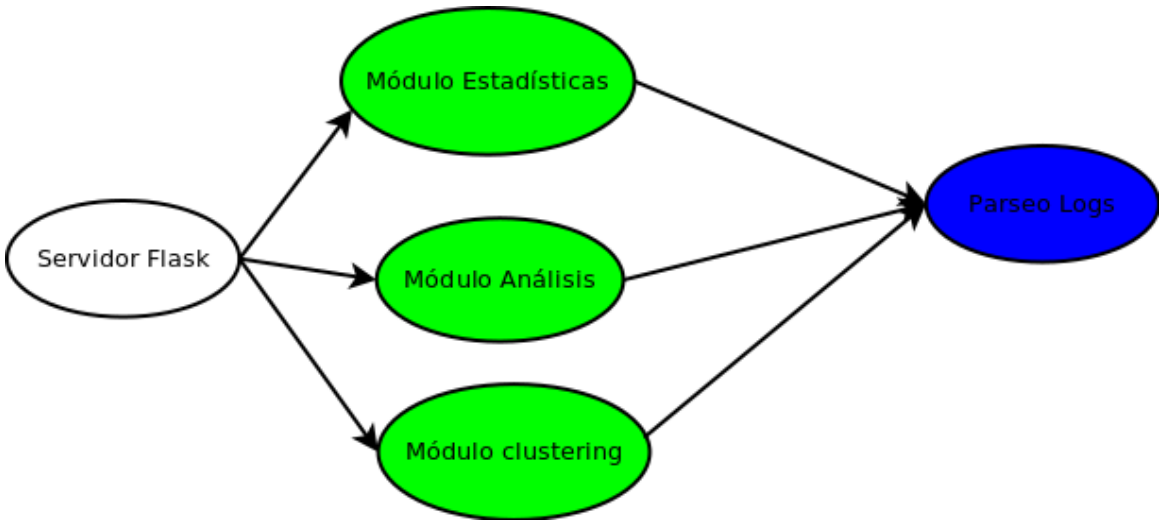


Figura 4.1: Interacción entre el servidor y el módulo de parsear *logs*

[1]Datos facilitados en 2008 por Andy Rubin, co-fundador de Android

- **Servidor-Base de datos**

  El servidor interactúa con la base de datos para validar el usuario introducido en el inicio de sesión, comprobando que se corresponde con el administrador del sistema, tal y como muestra la figura 4.2.

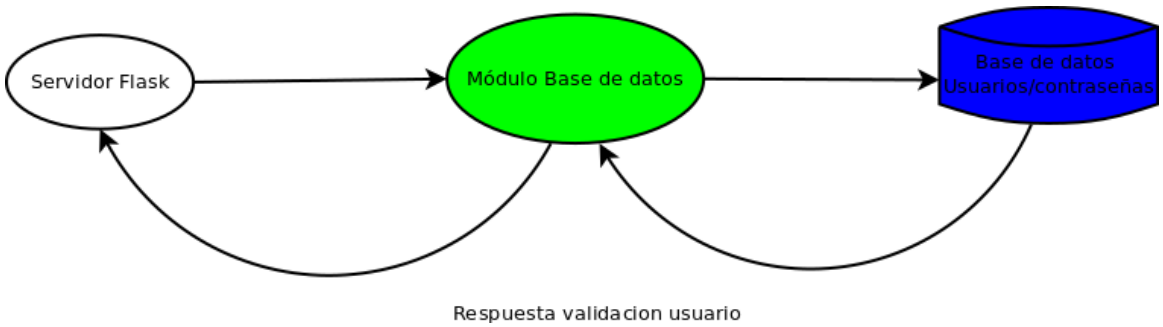


Figura 4.2: Interacción entre el servidor y la base de datos

Finalmente, el siguiente diagrama 4.3 muestra la interacción completa entre todos los componentes del sistema:

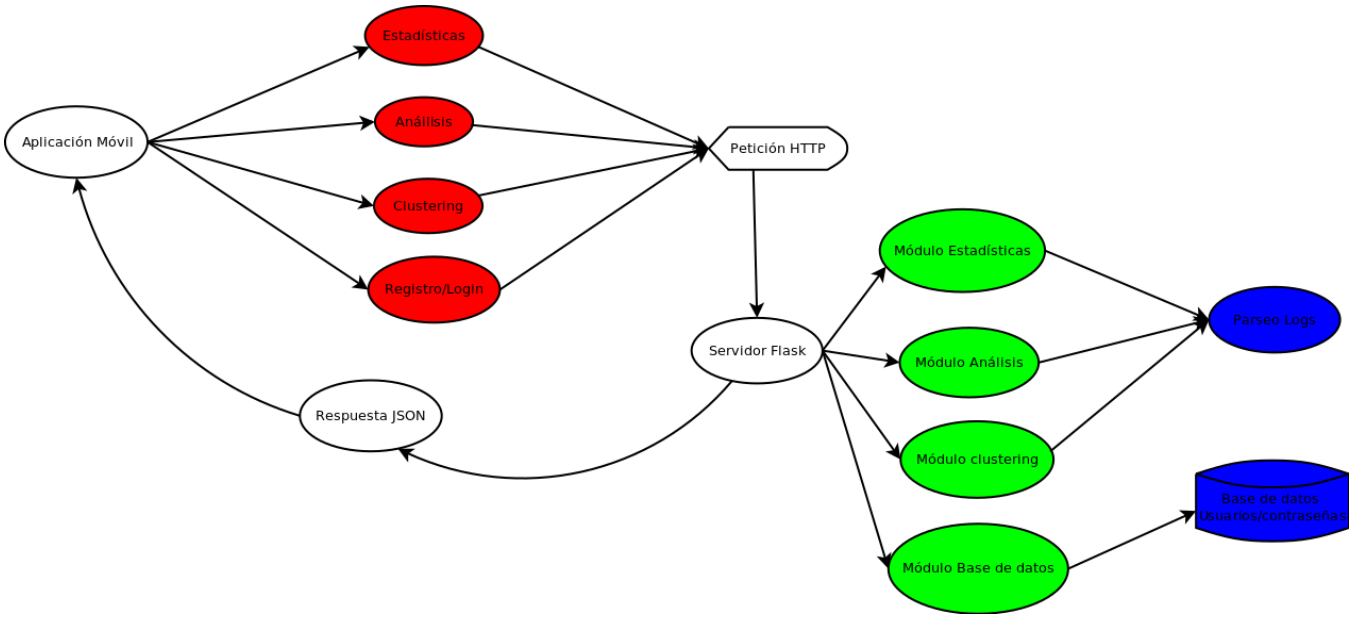


Figura 4.3: Diseño del sistema completa

La figura 4.4 muestra las maquetas de la aplicación Móvil.

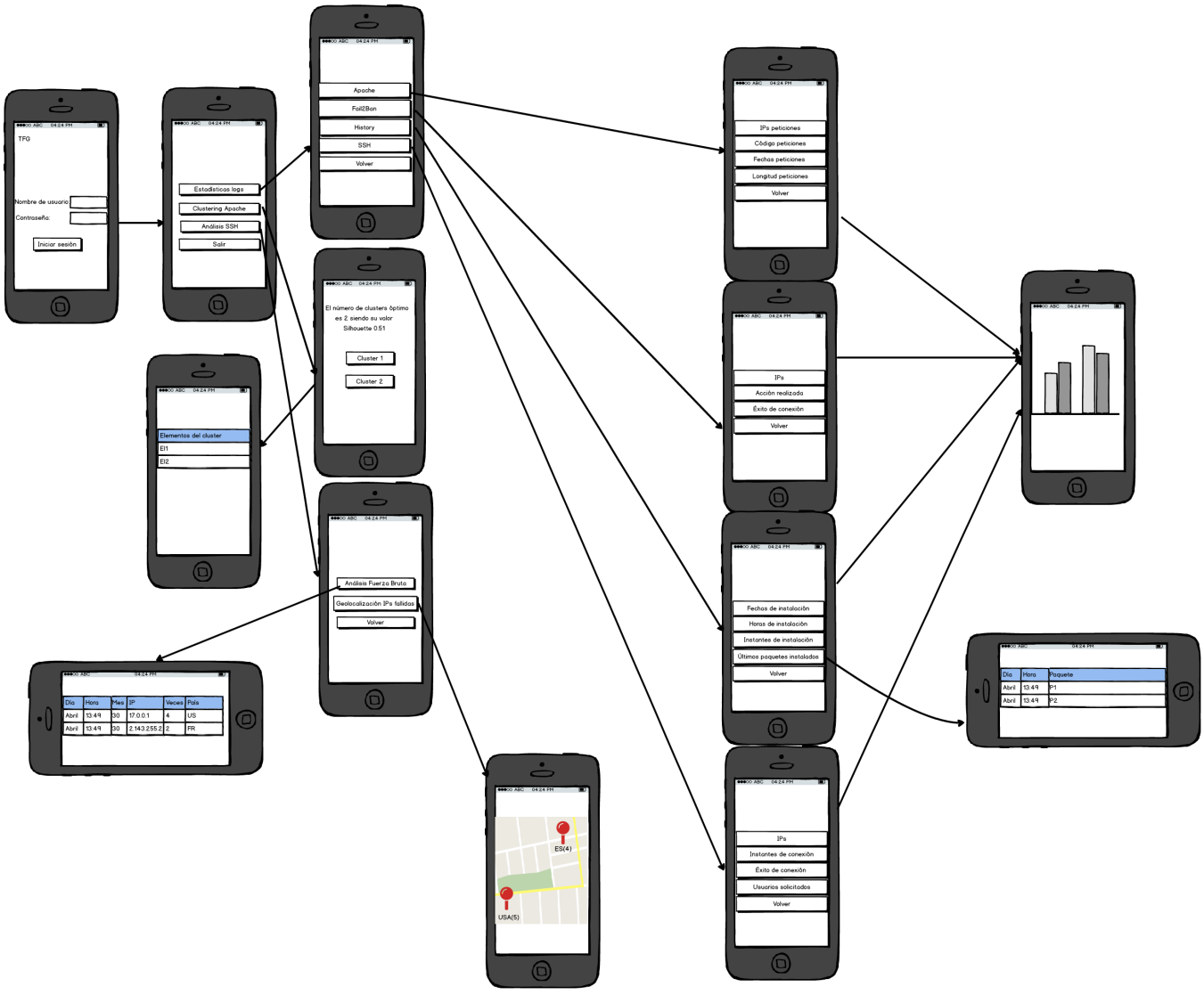


Figura 4.4: Maqueta de la aplicación Móvil

5

Desarrollo e implementación

Esta sección trata los detalles de la implementación y codificación del sistema completo, esto es, tanto las características del equipo que se ha usado para desarrollar la aplicación, como la estructuración de módulos y bibliotecas usadas en el proyecto.

5.1. Equipo de desarrollo

A continuación se exponen las características tanto hardware como software del equipo sobre el cual se ha llevado a cabo el desarrollo de la aplicación.

Hardware

El proyecto se ha desarrollado en un equipo con las siguientes características:

Equipo sobremesa Acer de 32-bits, con disco duro de 500 GB, sistema operativo Ubuntu 15.04 y procesador Intel Core I5-2467M de 1.60GHz x 4.

Software

Para la codificación e implementación del proyecto, se ha usado el siguiente software:

Software de programación

- Android Studio para la codificación de la aplicación Móvil.
- RStudio para la implementación de *clustering*.
- Control de versiones mediante BitBucket y Git en Linux.
- MySql Workbench para ver los contenidos de la base de datos.

Software de edición

- Día.
- Sublime Text para edición de texto.
- Gimp para edición de imágenes.

5.2. Plataformas y lenguajes de programación

A continuación se nombran y se explican los lenguajes de programación y plataformas usadas en el proyecto:

Python

Python es un lenguaje de programación interpretado y multiparadigma, que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje que usa tipado dinámico y es multiplataforma.

Toda la implementación del servidor se ha realizado en Python, así como los módulos referentes a análisis y estadísticas de *logs*.

Flask

Flask es un framework minimalista escrito en Python y basado en la especificación WSGI de Werkzeug y en el motor de templates Jinja2.

El servidor de la aplicación está basado en Flask. Cada ruta al servidor se asigna a una función en el programa que ejecuta Flask, y que a su vez llama a funciones de otros módulos de Python. Así, cuando la aplicación Móvil solicita cierta información de algún *log*, ésta manda la petición HTTP al servidor en Flask, y éste ejecuta la función adecuada del módulo correspondiente.

MySQL

MySQL es un sistema de gestión de bases de datos relacional, y considerado como una de las bases de datos open source más populares, sobre todo para entornos de desarrollo web.

La base de datos de usuarios, que contiene tanto los nombres de usuarios como sus contraseñas está implementada en MySql. Para la conexión del servidor con la base de datos, Flask usa librerías MySql de Python.

XML

XML (eXtensible Markup Language), es un lenguaje de marcas desarrollado utilizado para almacenar datos en forma legible. Su utilidad reside en que permite estructurar documentos grandes, además de dar soporte a bases de datos, y ser compatible entre sistemas para compartir la información de una manera segura, fiable y fácil.

R

R es un entorno y lenguaje de programación con un enfoque al análisis estadístico. Es una implementación de software libre. Además se trata de uno de los lenguajes más utilizados en investigación por la comunidad estadística, siendo muy popular en el campo de la minería de datos o las matemáticas financieras. También admite la posibilidad de cargar diferentes bibliotecas o paquetes con funcionalidades de cálculo o graficación.

Java

Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos, diseñado específicamente para tener pocas dependencias de implementación. Permite que los

desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo, basándose en la idea de que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

La parte cliente, que se corresponde a la aplicación Móvil Android, está desarrollada en su totalidad en Java.

Librería Volley

La librería Volley es una librería de Java que permite establecer conexión con un servidor web. En este caso, el lado cliente Android hace uso de dicha biblioteca para mandar las peticiones al servidor Flask. La devolución de resultados del servidor se hace en formato JSON, lo que facilita su procesamiento en el lado cliente.

Librería AndroidChartMP

La librería AndroidChartMP es una librería en Android para la representación gráfica de todo tipo de estadísticas, porcentajes y resultados fruto de un análisis previo de un conjunto de datos.

En este proyecto, AndroidChartMP se ha usado para representar las estadísticas de campos de *logs*. Así por ejemplo, se pueden ver las primeras N IPs que más peticiones SSH hayan hecho, siendo N un número fijado por el usuario previamente.

5.3. Estructuración del código

A continuación se detallan tanto las clases Java usadas para la implementación de la aplicación Móvil, como los módulos Python usados para la codificación del servidor y de los principales servicios que éste suministra.

5.3.1. Aplicacion Móvil Android

La estructuración de la aplicación se ha hecho en **paquetes** Java, estando así las clases que comparten la misma funcionalidad en el mismo paquete. Para la nomenclatura de los paquetes, se ha usado la nomenclatura de dominio inverso, empezando por *es.uam.eps.mellouk_tfg*. A continuación se explican los paquetes así como las clases más importantes del proyecto:

- **es.uam.eps.mellouk_tfg.connection:** Es el paquete que sirve para la conexión entre la aplicación Móvil y el servidor, y contiene la siguiente clase:
  - **ServerInterface.java**: Es la clase que implementa los métodos para establecer la conexión usando la librería Java *Volley*. Se compone de distintos métodos a llamar dependiendo de la petición que se quiera hacer al servidor, usando la llamada a *JsonArrayRequest*.
- **es.uam.eps.mellouk_tfg.identification:** Es el paquete que sirve para la identificación de los usuarios en la aplicación, y contiene la siguiente clase:
  - **Login.java**: Clase que contiene los métodos para la identificación de un usuario en el sistema. El nombre de usuario y la contraseña se mandan al servidor comprobando la existencia del usuario en su base de datos, para que en su caso devolver un mensaje de aceptación y poder acceder a la aplicación.

- **es.uam.eps.mellouk_tfg.SSH:** Es el paquete que contiene las clases responsables del análisis del *log* de SSH, es decir, que permiten tanto la geolocalización de IPs como análisis de intentos de ataques por fuerza bruta, y son dos:

  - **Geolocation.java**: Clase que implementa los métodos para poder representar gráficamente en un mapa las IPs fallidas del *log* SSH. Hace uso de las librerías de GoogleMaps.
  - **ForceBrute.java**: Clase que se encarga de la obtención y representación en forma de tabla de las IPs desde las cuales hay más probabilidades de haber realizado un ataque por fuerza bruta. Después de su obtención, representa la fecha, la IP, el número de intentos fallidos que ha realizado la IP, así como el país al cual pertenece.

- **es.uam.eps.mellouk_tfg.statistics:** Es un paquete que contiene una única clase encargada de las estadísticas de *logs*:

  - **StatisticsHistogram**: Clase que representa el histograma estadístico de los *logs* mediante la librería AndroidChartMp. Dependiendo del nombre del archivo y del campo pasado a la actividad, ésta representa las estadísticas del campo correspondiente a un *log* u a otro.

- **es.uam.eps.mellouk_tfg.menus:** Este paquete contiene diversas clases encargadas de mostrar los menús y pantallas de la aplicación.

- **es.uam.eps.mellouk_tfg.preferences:** Paquete con clases que gestionan las preferencias de usuario. En este caso se gestionan básicamente el número tope de estadísticas, N, que el usuario elige para mostrar los primeros N resultados de un determinado campo.

- **es.uam.eps.mellouk_tfg.history:** Paquete encargado de mostrar los últimos paquetes instalados, resultado del *log* History.

- **es.uam.eps.mellouk_tfg.clustering:** Paquete que contiene a las clases que tienen como funcionalidad la agrupación de los datos de peticiones Apache.

5.3.2. Módulos Python

- **Server.py**: Es el módulo Flask encargado de inicializar el servidor en un determinado puerto y quedarse en escucha. Este módulo contiene funciones de otros módulos a las que hay que llamar dependiendo del contenido de la petición HTTP que llegue al servidor. Por ejemplo al mandar la aplicación Android la petición:

  *http://192.168.1.111:5008/statistics?*log*=APACHE&field=IP&top=5*

  Se ejecutaría la siguiente función:

  ```
  @app.route('/statistics',methods=['GET', 'POST'])
  getStatisticsTop()
  ```

  que llamaría a *jsonStatistics* del módulo *statistics*, y que en este caso devolvería las estadísticas de las 5 IPs más solicitadas en el *log* Apache en formato **JSON**.

- **Statistics.py**: Es el módulo encargado de procesar las estadísticas generales de los *logs*. Dado el nombre de un *log* y un campo suyo, devuelve un top de estadísticas pedidas por el usuario. Después de leer los ficheros XML correspondientes al *log* y campo en cuestión mediante la función *getDictionary(*log,*field)*, se ordenan y se seleccionan las N primeras mediante la función *sorted*, para finalmente mandarse en formato **JSON** usando la librería json de Python.

- **sshAnalyze.py**: Módulo encargado de analizar el *log* correspondiente a SSH. una vez *parseado* el fichero SSH correspondiente, su utilidad se resume en dos funciones:

  - Analizar todas las peticiones que figuran en el fichero separándolas por franjas horarias, y seleccionando aquellas que tienen riesgo de ser ataques por fuerza bruta.
  - Agrupar las IPs fallidas por países y mandar su valor, país, latitud y longitud al servidor en caso de pedir geolocalización. La inclusión de los valores latitud y longitud se debe a su necesidad para la representación del país en GoogleMaps.

  Este módulo es uno de los más complejos, debido a la gran cantidad de datos que tiene que agrupar y analizar.

- **Clustering.py**: Módulo Python para hacer *clustering* sobre los campos de petición de HTTP y hacer las agrupaciones pertinentes. Para ello, se usa la librería de R rpy2.

5.3.3. Módulos de *parseo*

Para los ficheros de parseo, cada fichero *log*, digamos *log*.txt, tiene su propio módulo *log*.py, que usando la librería logparser, genera el correspondiente fichero log.xml *parseado*.

6
Plan de Pruebas

En este capítulo se describe el plan de pruebas llevado a cabo durante el desarrollo y después de la implementación del proyecto. Un plan de pruebas es la especificación de lo que se desea probar y de cómo ejecutar dichas pruebas. Las pruebas y el diseño de un buen plan de pruebas son de gran importancia para garantizar la calidad de los programas, y se convierte en una tarea de vital importancia para detectar errores en tempranas etapas de desarrollo.

La fase de pruebas es una fase fundamental en el proceso de desarrollo del software cuyo objetivo principal es dar con todos los errores posibles antes de dar el proyecto por acabado. Se han realizado dos tipos de pruebas, las pruebas unitarias y las pruebas de integración.

Las **pruebas unitarias** se ejecutan en el entorno de trabajo y son una forma de comprobar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado, verificando que cada módulo soporte la entrada de datos erróneos o extremos.

Las **pruebas de integración** son aquellas que se realizan una vez que se han aprobado las pruebas unitarias. Es la fase de la prueba de software en la cual módulos individuales de software son combinados y probados como un grupo. Son las pruebas posteriores a las pruebas unitarias y preceden a las pruebas del sistema.

6.1. Pruebas unitarias

Tanto para el servidor Flask como para los módulos de *parseo* se han realizado pruebas unitarias para comprobar el correcto funcionamiento individual. A continuación se detalla el plan de pruebas realizado para ello.

Módulos de parseo

Para los módulos de *parseo*, se ha empleado para cada *log*, digamos logX, un módulo logX-ParserTest.py, en el cual se prueba la funcionalidad. Se le pasa a cada uno líneas de un fichero de su *log* correspondiente de prueba, y se comprueba que la salida XML generada coincide con la esperada. Se han empleado los siguientes módulos para poder llevarlo a cabo:

- ApacheParserTest.py

- SSHParserTest.py
- HistoryParserTest.py
- FailBanParserTest.py
- UfwParserTest.py

Todos los ficheros anteriores son llamados por el *script* TestsParser.py, que comprueba la funcionalidad de la totalidad del módulo de *parseo* para todos los *logs*.

Módulos de servidor Flask

clusteringTest.py: comprueba que dado un fichero XML de Apache, el resultado del clustering es el esperado para eso conjunto de datos.

sshGeoTest.py: Dado un fichero XML de SSH, este módulo, comprueba que el número de IPs fallidas para cada país coincide con el calculado previamente, para su posterior representación y geolocalización. Más concretamente, comprueba que el resultado en JSON es el correcto.

sshTableTest.py: Módulo que comprueba para un conjunto de ficheros XML de SSH, la coincidencia entre los resultados esperados y obtenidos de la función que calcula por cada IP fallida, el número de intentos de conexión realizado por fecha, indicando asimismo su país. Se comprueba que la salida JSON contiene los datos correctos, que serán representados por la aplicación Móvil en forma de tabla.

En cuanto a las estadísticas, para cada campo de cada *log*, se ha creado un fichero de prueba para contrastar que el resultado del top de estadísticas obtenido es el reflejado realmente en los *logs*.

6.2. Pruebas de integración

En las pruebas de integración se prueba la correcta interacción entre los distintos componentes que han sido probados individualmente en las pruebas unitarias, así, se comprueba la funcionalidad del sistema completo.

- **Estadísticas *logs***: Para comprobar las estadísticas de los *logs*, se ha ejecutado la aplicación Móvil con conexión al servidor, comprobando que los resultados obtenidos gráficamente se corresponden con aquellos devueltos por el servidor en las pruebas unitarias para cada *log*.
- **Análisis fuerza bruta SSH**: Se ha comprobado que el módulo de prueba unitaria para el análisis de fuerza bruta se corresponde con la tabla mostrada por la aplicación Móvil. Asimismo, se ha simulado un ataque por fuerza bruta mediante el software Hydra[1]. El resultado ha sido satisfactorio y se ha mostrado tanto en la tabla de análisis SSH como en la geolocalización de la IP del ataque.
- **Geolocalización IPs fallidas**: En cuanto a la geolocalización de IPs fallidas, se ha corroborado la coincidencia entre los países junto con el número de intentos fallidos en cada país, y el resultado de la prueba unitaria de Python. Así, los países se han colocado en la ubicación del mapa correcta.
- **Clustering Apache**: Al igual que en los casos anteriores, la finalidad de la prueba ha sido comprobar que el resultado de los *clusters* visualizado y mostrado por la aplicación Móvil es el esperado, y el devuelto por el servidor.

[1] https://hackertarget.com/brute-forcing-passwords-with-ncrack-hydra-and-medusa/

Mediante las anteriores pruebas, ha quedado claro que los datos expuestos por la aplicación Móvil coinciden con los datos que ha obtenido el servidor después de su análisis de los *logs*, y por lo tanto afirman el correcto funcionamiento entre los distintos módulos de la aplicación, en este caso entre la aplicación Móvil y el servidor Flask.

7

Conclusiones y trabajos futuros

Con el presente proyecto se pretendía afrontar un caso práctico en el que aplicar de modo integral conocimientos adquiridos durante el doble grado en Informática y Matemáticas. A lo largo de cinco años de estudio, siempre he sentido curiosidad e interés por las asignaturas de Matemáticas relacionadas con la criptografía y la seguridad, hecho que me ha motivado al desarrollo un Trabajo de Fin de Grado que manifieste los aspectos prácticos de la seguridad en el campo de la informática, tomando como hilo conductor el análisis de *logs*.

Teniendo en cuenta lo anterior, se ha pensado llevar a cabo un proyecto orientado al desarrollo de una herramienta que permita unificar y analizar un conjunto de *logs* del sistema Linux. Esta necesidad surge debido a la no existencia de un estándar de *logs*, que permita extender un modelo de análisis y recogida a todos los *logs* de los que dispone el sistema. La detección de los campos importantes de algunos *logs* tales como Apache, SSH o los cortafuegos, representarlos gráficamente, así como hacer un análisis posterior de los resultados es uno de los objetivos que ha pretendido cubrir este trabajo, con la motivación de poder encontrar y detectar comportamientos anómalos en el sistema.

Además se ha dado una gran importancia a la interfaz gráfica, usando una aplicación Móvil, siendo así una manera muy cómoda e intuitiva para mostrar los resultados.

La tarea de pasar de un conjunto de ficheros *log*, con información heterogénea y difícil de procesar en primera instancia, a unos resultados que son interpretables a simple vista mediante gráficos y tablas, representa uno de los alcances principales del proyecto que lo hace útil e interesante para un administrador del sistema.

El estudio realizado para llevar a cabo el proyecto ha sido plasmado a lo largo de todo el presente documento. En primer lugar se ha hecho un primer trabajo de estudio de los *logs*, el cual ha estado orientado a la detección de los campos que más interés puedan tener a la hora de la identificación de anomalías. Tras la identificación se llevó a término el *parseo* mediante expresiones regulares de los *logs*, que derivó en la generación de los ficheros XML correspondientes. Un segundo paso ha sido detallar el procedimiento llevado a cabo para el análisis de ataques SSH por fuerza bruta, así como la implementación y validación de *clusters* para las peticiones HTTP.

Por último, se especifica el plan de pruebas realizado para verificar el correcto funcionamiento de todas las características del sistema. Así pues, se puede concluir que el proyecto desarrollado

cumple con los requisitos inicialmente planteados.

Cabe mencionar que una de las mayores dificultades encontradas es el tratamiento de la gran cantidad de información que contienen los *logs*, sabiendo cuál es realmente la importante, así como su posterior *parseo*, debido a que no existe ningún estándar común para formatos de *logs*. Por otra parte, al usar en la aplicación varios lenguajes y plataformas de programación (Python, R, Java, Mysql, XML, Android), ha habido un gran proceso de aprendizaje de la mayoría de esos lenguajes, como Python, XML, R y Mysql. Es de especial relevancia el tiempo invertido en la comprensión del uso de expresiones regulares en Python, un campo que nunca había tocando anteriormente y en el que tuve que formarme al respecto para poder incluirlo en la parte de *parseo*. Además, el uso de tantos lenguajes ha implicado ciertas dificultades a la hora de las configuraciones y de la interacción entre ellos, por lo que la integración entre los distintos módulos para el correcto funcionamiento de la aplicación ha sido algo costosa.

Así pues, la herramienta diseñada e implementada resulta ser una primera aproximación a sistemas reales. Estos sistemas son complejos tanto en elementos hardware como software, de forma que su correcto tratamiento involucra el uso de multitud de lenguajes de programación y de un elevado número de configuraciones para la correcta integración.

Como trabajos futuros, se propone la inclusión de la opción de registro de usuario, pudiendo así haber varios administradores del sistema. Por otra parte, en este proyecto se han tratado simplemente un subconjunto de *logs*, ya que la idea es dar un primer paso hacia la herramienta y por lo tanto, se pueden añadir en un futuro más *logs* si se desea oportuno para su análisis. Gracias a la modularidad propuesta, añadiendo el módulo de *parseo* correspondiente, así como la clase de visualización Java pertinente para la aplicación Móvil, se tendría un *log* más en la aplicación.

Por otra parte, la inclusión de correlación entre distintos *logs* es uno de los trabajos futuros que mejoraría la precisión de los resultados y de la detección de anomalías en el sistema. No obstante,lo más relevante es un módulo de procesamiento y búsqueda de anomalías, dado que por ahora la herramienta sólo permite el análisis de *logs* mediante histogramas y *clustering*.

Glosario

clustering procedimiento de agrupación de un conjunto de datos de acuerdo con un criterio, que por lo general suelen ser la distancia o similitud. 20

ACL *Access Control List*, forma de determinar los permisos de acceso apropiados a un determinado objeto, para fomentar la separación de privilegios. 5

Cisco IOS *Internetwork Operating System*, software utilizado en la gran mayoría de routers de Cisco Systems. Se compone de un paquete de funciones de enrutamiento y conmutamiento.. 5

CSV *Comma-Separated Values*, tipo de documento en formato abierto. 12

DoS *Denial of Service*, ataque a un sistema de computadoras o red que causa que un servicio o recurso sea inaccesible a los usuarios legítimos. 9

hash función computable mediante un algoritmo, que tiene como entrada un conjunto de elementos, que suelen ser cadenas, y los convierte en un rango de salida finito, normalmente cadenas de longitud fija. 10

HTTP *Hypertext Transfer Protocol*, el protocolo de comunicación que permite las transferencias de información en la World Wide Web. Es un protocolo sin estado, usando las cookies, que es información que un servidor puede almacenar en el sistema cliente.. 27

Inyección SQL método de infiltración de código intruso que se vale de una vulnerabilidad informática presente en una aplicación en el nivel de validación de las entradas para realizar consultas a una base de datos.. 20

IP *Internet Protocol*, protocolo de comunicación de datos digitales clasificado funcionalmente en la capa de red según el modelo internacional OSI.. 12

JSON *JavaScript Object Notation*, formato de texto ligero para el intercambio de datos, considerado un subconjunto de la notación literal de objetos de JavaScript. 33

MD5 *Message-Digest Algorithm 5*, algoritmo de reducción criptográfico de 128 bits ampliamente usado, y diseñados por el profesor Ronald Rivest del MIT. 25

MIT *Massachusetts Institute of Technology*. 11

MVC *Modelo Vista Controlador*. 27

NFA *Nondeterministic Finite Automatons*. 9

Postfix servidor de correo de software libre y código abierto, para el enrutamiento y envío de correo electrónico creado para ser una alternativa más rápida, fácil de administrar y segura Sendmail. 17

Qmail Servidor de correo electrónico (SMTP) hecho para Unix. 17

RAM *Random Access Memory*, memoria de trabajo de computadoras para el sistema operativo, los programas y la mayor parte del software, donde se cargan todas las instrucciones que ejecuta la unidad central de procesamiento (procesador) y otras unidades del computador. 28

ReDoS *Regex Denial of Service*, ataque de denegación de servicio en la validacio?n de Expresiones Regulares, introduciendo algunas de ellas que llevan demasiado tiempo en ser evaluadas.Ataque ReDos, ataque de denegación de servicio en la validacio?n de Expresiones Regulares, introduciendo algunas de ellas que llevan demasiado tiempo en ser evaluadas.. 9

rpy2 Librería de Python para el manejo de código R. 35

SEC *Simple Event Correlator*, herramienta diseñada para tareas de correlaciones de eventos en el ámbito de seguridad de sistemas y de redes.. 12

SIEM *Security Information and Event Management*. 1

Silhouette Método de interpretación y validación de la consistencia de clusters. Es un valor de medida de cuánto de similar es un objeto a su propio cluster (cohesion) y también comparado con otros clusters (separación). 22

SSH *Secure SHell*, nombre de un protocolo y del programa que lo implementa, que sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos. 28

Ubuntu sistema operativo basado en GNU/Linux y que se distribuye como software libre. 7

UFW *Uncomplicated Firewall*, cortafuegos diseñado para ser de fácil uso desarrollado por Ubuntu. Utiliza la línea de comandos para configurar las iptables usando un pequeño número de comandos simples. 19

Unix sistema operativo portable, multitarea y multiusuario desarrollado en 1969 en los laboratorios Bell. 7, 15

vsftpd *Very Secure FTP Daemon*, servidor FTP sistemas Unix. 17

WSGI *Web Server Gateway Interface*, especificación para una interfaz simple y universal entre servidores y aplicaciones web, o frameworks, para el lenguaje de programación Python. 32

XML *eXtensible Markup Language*, lenguaje de marcas desarrollado por el World Wide Web Consortium (W3C) utilizado para almacenar datos en forma legible. 12, 35, 37, 38, 42

Bibliografía

[1] A. A. Chuvakin, *Logging and Log Management: The Authoritative Guide to Understanding the Concepts Surrounding Logging and Log Management. 1st Edition.* Syngress, 2012 (vid. págs. 1, 5, 6).

[2] B. Schneier y J. Kelsey, "Secure audit logs to support computer forensics", 2006 (vid. pág. 1).

[3] S. H. Huseby, *Innocent Code: A Security Wake-Up Call for Web Programmers.* Wiley; 1 edition, 2004 (vid. págs. 5, 19).

[4] J. Luby. (2014). Data privacy, anonymization and log data, dirección: https://jluby.blogs.balabit.com/2014/09/10/data-privacy-anonymization-and-log-data/ (visitado 23-06-2016) (vid. pág. 11).

[5] B. Barett. (2013). MIT teaching AI2 how to help stop cyberattacks., dirección: https://www.wi%20red.com/2016/04/mits-teaching-ai-help-analysts-stop-cyberattacks (visitado 02-06-2016) (vid. pág. 11).

[6] 2bits. (2014). Identifying aggressive crawlers using go access., dirección: http://2bits.com/apache/identifying-aggressive-crawlers-using-goaccess.html (visitado 20-05-2016) (vid. pág. 11).

[7] R. Vaarandi. (2015). Sec manual., dirección: https://simple-evcorr.github.io/man.html (visitado 08-05-2016) (vid. pág. 12).

[8] J. Turnbull, *The Logstash Book.* Kindle Edition, 2013 (vid. pág. 12).

[9] Splunk. (2014). Splunk web page, dirección: http://www.splunk.com/ (visitado 10-06-2016) (vid. pág. 12).

[10] logtrust. (2013). Logtrust web page, dirección: https://www.logtrust.com (visitado 10-06-2016) (vid. pág. 12).

[11] Papertrail. (2012). Papertrail web page, dirección: https://papertrailapp.com/ (visitado 10-06-2016) (vid. pág. 12).

[12] J. F. Kurose y W. Ross, *Computer Networking: A Top-Down Approach.* Pearson 6th edition, 2012 (vid. págs. 17, 18).

[13] OWASP. (2016). Sql injection, dirección: https://www.owasp.org/index.php/SQL_Injection (visitado 20-06-2016) (vid. pág. 19).

[14] C. Hennig y M. Meila, *Handbook of Cluster Analysis.* Chapman y Hall/CRC, 2015 (vid. pág. 20).

[15] Wikispace. (2016). Manhattan distance, dirección: https://heuristicswiki.wikispaces.com/Manhattan+Distance (visitado 15-04-2016) (vid. pág. 21).

[16] F. P. Miller, *Levenshtein Distance.* VDM Publishing, 2009 (vid. pág. 21).

[17] C. C. Aggarwal, *Handbook of Cluster Analysis.* Chapman y Hall/CRC, 2014 (vid. pág. 21).

[18] C. C. Aggarwal y C. K. Reddy, *Data Clustering: Algorithms and Applications.* Chapman y Hall/CRC, 2013 (vid. pág. 21).

[19] V. Kumar. (2009). The k-means algorithm, dirección: http://web.khu.ac.kr/~tskim/PatternClass%20Lec%20Note%2020-2%20Kmeans%20PT.pdf (visitado 20-02-2016) (vid. pág. 21).

[20] Z. Huang, *Extensions to the k-Means Algorithm for Clustering Large Data Sets with Categorical Values*. Kluwer Academic Publishers, 1998 (vid. pág. 21).

[21] C. Torrano Gimenez, "Study of stochastic and machine learning techniques for anomaly-based web attack detection", PhD Thesis, Universidad Carlos III de Madrid Escuela Politecnica Superior, 2015 (vid. págs. 21, 22).

[22] Y. Zhang y Borror. (2004). Robustness of the markov-chain model for cyber-attack detection, dirección: http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=%201282169&abstractAccess=no&userType=inst (visitado 16-04-2016) (vid. pág. 21).

[23] O. Kolesnikov y W. Lee. (2006). Polymorphic blending attacks. in proceedings of the 15th conference on usenix security symposium, volume 15, dirección: http://dl.%20acm.org/citation.cfm?id=1267336.1267353 (visitado 12-03-2016) (vid. pág. 22).

[24] A. L. N. Fred, F. Jorge y F. Duarte, "On consensus clustering validation", Bachelor Thesis, Instituto Superior Técnico, Lisboa, Portugal, 2015 (vid. pág. 22).

[25] E. Wilding, "Computer forensics, trends and concern", págs. 15-18, 1997.

[26] B. McCarty, *Learning Debian GNU/Linux*. O'Reilly Media, 1999.

[27] R. Smith, *Linux Administrator Street Smarts: A Real World Guide to Linux Certification Skills*. Sybex, 2006.

[28] F. Lopez y V. Romero, *Mastering Python Regular Expressions*. Packt, 2014.

[29] J. Kuhnel, "Centralized and structured log file analysis with open source and free software tools", 2013.

[30] *Android developers site*, https://developer.android.com/index.html.

[31] D. J. Barrett, R. E. Silverman y R. G. Byrnes, *SSH, The Secure Shell: The Definitive Guide*. O'Reilly Media, 2005.

[32] C. Aulds, *Linux Apache Web Server Administration, Second Edition*. Sybex; 2002.

[33] *Wikipedia: silhouette*, https://en.wikipedia.org/wiki/Silhouette_(clustering).

[34] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, 2014.

A

Jerarquía de directorios

Para el correcto funcionamiento del servidor, éste tiene que analizar y trabajar sobre los ficheros *log* ya parseados. Por lo tanto, tiene que haber una jerarquía de separación para diferenciar las labores del *parseo* de las de análisis y obtención de estadísticas que realiza el servidor.

Para ello, se ha diseñado la siguiente jerarquía de directorios, que se ilustra en la figura A.1:

- **TFG**: Es el directorio principal, que contiene todos los ficheros necesarios.
  - **logparser**: Directorio con todos los ficheros para las operaciones de *parseo* de logs.
    - **parsers**: Directorio con el conjunto de ficheros Python que parsean los ficheros *log* a formato XML.
    - **XMLResults**: Directorio donde se guardan los ficheros XML resultantes del parseo.
    - **logs**: Directorio que contiene los ficheros logs sin tratamiento. Es la ubicación de la cual se basan los parsers para realizar la conversión a XML.
    - scriptParseo.sh: Script en bash con la finalidad de actualizar los *logs* y parsearlos con la frecuencia de un día, para así mantenerse actualizados.
  - **loganalyze**: Directorio donde se encuentran todos los ficheros necesarios para la ejecución del análisis del servidor.
    - **PythonServer**: Directorio que aloja el servidor propiamente dicho.
    - Server.py: Fichero Python que ejecuta el servidor Flask.

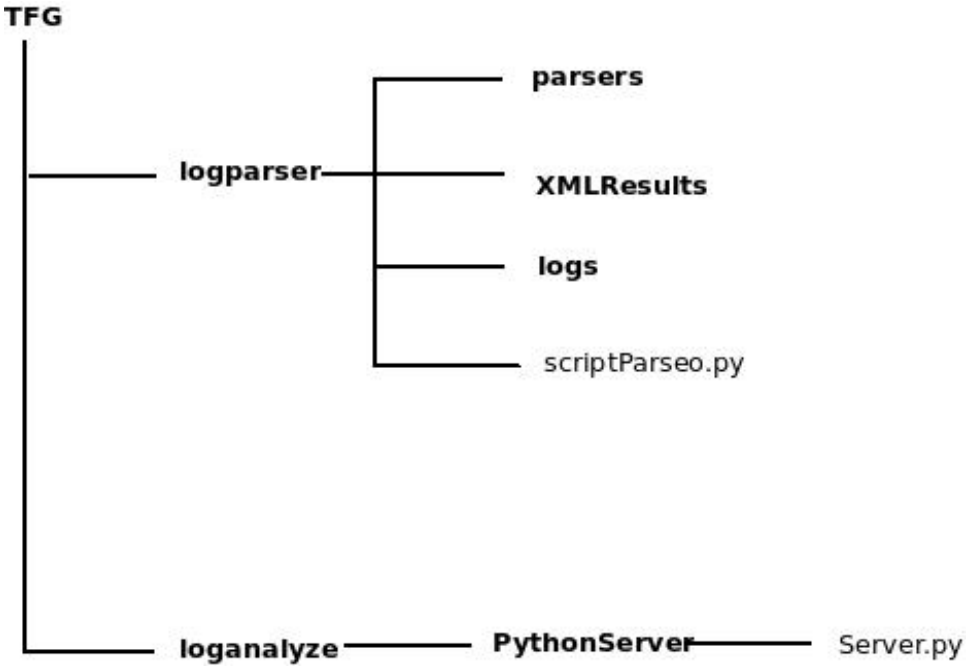


Figura A.1: Jerarquía de directorios del servidor

B

Actualización de ficheros *log*

El servidor analiza los *logs* que se encuentran en el directorio */logs*, por lo tanto tienen que mantenerse actualizados a medida de que el sistema los vaya generando.

Para ello, se ha hecho un *script* que se ejecuta una vez al día. De esta manera, la actualización de los *logs* se hace con una frecuencia razonable y adecuada, dada que los ficheros logs están cambiando constantemente a lo largo del día.

El *script*, denominado scriptParseo, se encuentra en la raíz del directorio principal *logparsers*, y actúa copiando los nuevos ficheros *log* del sistema al directorio */logs* y *parseando* cada uno, para así a la hora de pedir las estadísticas y el análisis, la versión sea actualizada.

```
# Script para actualización de logs

#!/bin/sh

cp /var/log/apt/history.log ./logs/history.txt

cp /var/log/apt/fail2ban.log ./logs/fail2ban.txt

cp /var/log/apache2/access.log ./logs/APACHE.txt

cp /var/log/auth.log ./logs/SSH.txt

python /parsers/fail2ban.py

python /parsers/APACHE.py

python /parsers/SSH.py

python /parsers/history.py
```

C

Configuración básica para UFW

El siguiente *script* define unas reglas básicas para la configuración inicial de cortafuegos de un servidor Web usando UFW, uno de los procesos vitales para la seguridad del servidor:

```
#!/bin/sh

# Desactivar el firewall
$ ufw disable


# Resetear todas las reglas
$ ufw reset


# Denegar el tráfico de entrada, y permitir el de salida
$ ufw default deny incoming
$ ufw default allow outgoing


# Abrir el puerto para el SSH
$ ufw allow ssh


# Abrir el puerto para la www
$ ufw allow www


# Abrir el puerto para el email
$ ufw allow 25/tcp


# Abrir el puerto para MySQL
$ ufw allow 3306/tcp


# Abrir el puerto para el protocolo ntpd
$ ufw allow ntp
```

```
# Activar el firewall
$ ufw enable

# Por último listar las reglas que hemos añadido
$ ufw status verbose
```

D

Planificación del proyecto

A continuación se detalla la planificación llevada a cabo para el seguimiento del proyecto:

- **Diciembre 2015**: Planteamiento del problema definiendo su alcance.
- **Enero 2016**: Análisis de requisitos. Se han definido los requisitos del proyecto, reconociendo requisitos incompletos, ambiguos o contradictorios, para así tener una visión global de la funcionalidad que va a tener el proyecto.
- **Febrero 2016**: Diseño de los componentes del sistema que dan respuesta a las funcionalidades descritas en la etapa de análisis. Definición de las entidades de negocio, en base a diagramas que permiten describir las distintas interacciones en el sistema.
- **Marzo 2016**: Selección de los lenguajes de programación adecuados para la realización del proyecto, después de un previo estudio de sus ventajas en el ámbito del proyecto actual.
- **Abril-Mayo 2016**: Codificación. Se ha llevado en este período la implementación de los módulos que integran el proyecto, esto es, tanto la aplicación Móvil Android como el servidor Python Flask.
- **Junio 2016**: Pruebas. Período de realización de pruebas unitarias para cada módulo, así como las pruebas de integración para verificar el correcto funcionamiento del proyecto.

E

Comandos para la instalación del servidor

Para la correcta compilación y funcionamiento del servidor, hace falta la instalación de un conjunto de paquetes y librerías que se especifican en el siguiente *script instalacion.sh*.

```
#!/bin/sh
#instalacion Python2.7
add-apt-repository ppa:fkrull/deadsnakes
apt-get update
apt-get install python2.7
#Instalación flask
pip install Flask
#Instalación MySql
apt-get install mysql-server
#Instalación librerías MySql para flask
pip install flask-mysql
#Instalación librerías hash
pip install Werkzeug
#Instalación librerías estadísticas Python
apt-get install python-scipy
pip install numpy
#Instalación librerías para localización IP Python
pip install python-geoip
pip install python-geoip-geolite2
pip install pygeoip
#Instalación base de datos de los países de las IPs
```

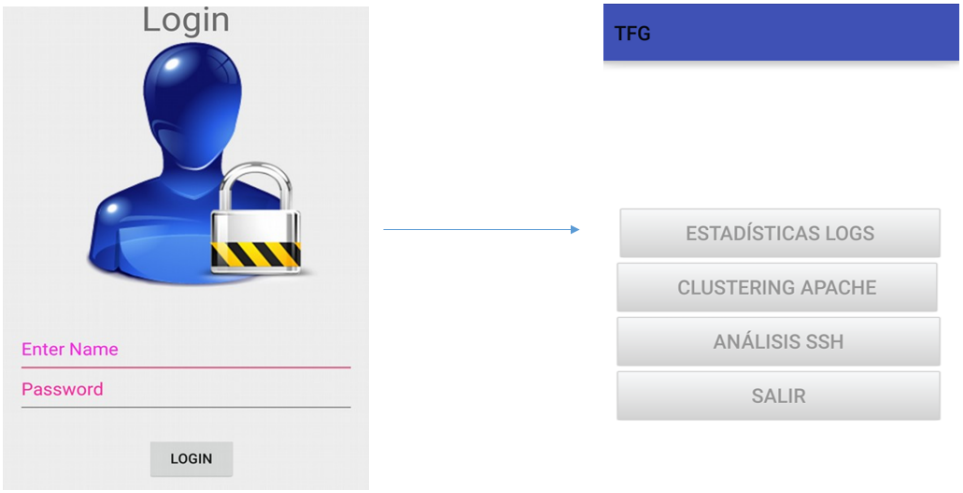
```
wget -N http://geolite.maxmind.com/download/geoip/database/GeoLiteCity.dat.gz
gunzip GeoLiteCity.dat.gz
mv GeoLiteCity.dat /usr/local/share/GeoIP/
#Instalación librería de R para Python
apt-get install libblas-dev liblapack-dev
pip install rpy2
```

F

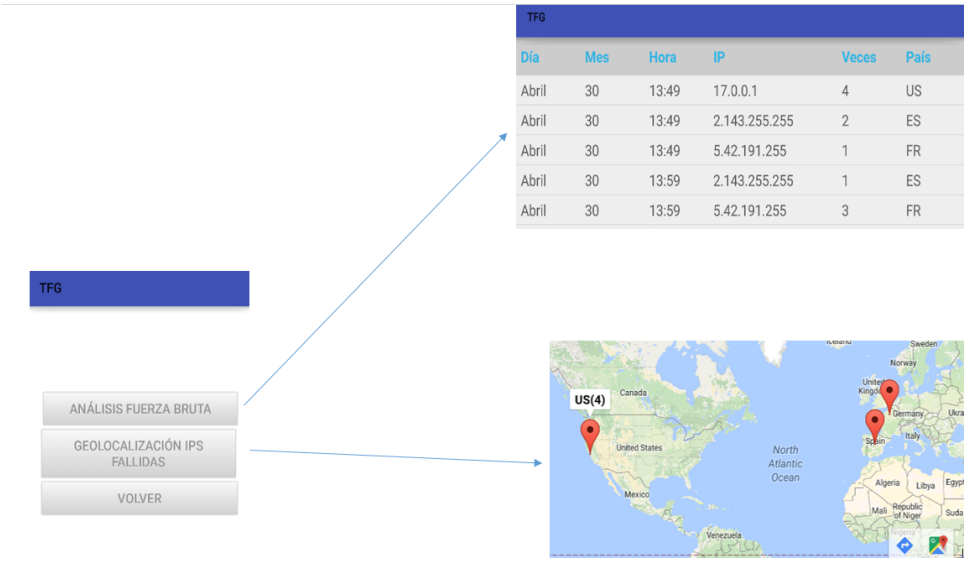
Capturas de la aplicación

A continuación se muestran algunas de las capturas más significativas de la aplicación:

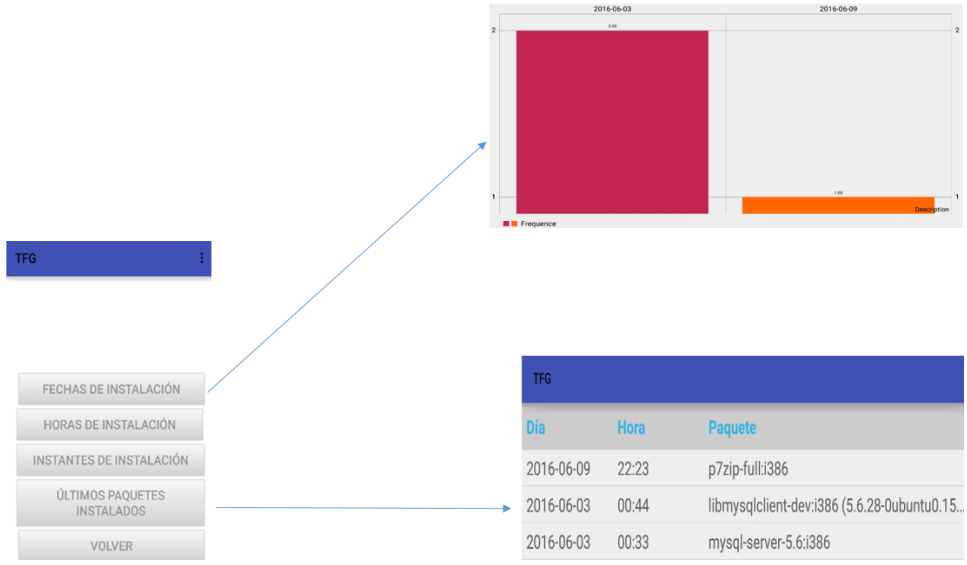
- **Inicio:**



- **Análisis SSH:**



- **Estadísticas de paquetes instalados:**



- **Estadísticas del estado de petición de Apache:**